Escuela Politécnica Superior

SwarmFlight

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:

Ángel Pérez Arroyo

Tutor/es:

Fidel Aznar Gregori

Universitat d'Alacant
Universidad de Alicante

Junio 2020

Índice de contenido

Índice de Ilustraciones

Preámbulo

Este proyecto ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades (Spain), proyecto RTI2018-096219-B-I00. Proyecto co-financiado con fondos FEDER

Resumen

En la actualidad el ámbito de los vehículos aéreos autónomos (drones) está en auge y tiene una aplicación práctica e incidencia en la actividad de muchas empresas. Pero su utilización se ha venido limitando a un solo vehículo con una única misión.

Mediante el presente Trabajo de Final de Grado se ha creado desde cero una aplicación para dispositivos Android (móviles y tabletas), que permite la comunicación con un enjambre de drones de la marca DJI desde un sólo cliente, de una manera sencilla e intuitiva para cualquier tipo de usuario.

Así la aplicación desarrollada permite que un grupo de drones (swarm drones) pueda realizar tareas de forma organizada, alcanzando una mayor eficacia y productividad.

DJI Mobile SDK es la herramienta esencial para poder establecer la comunicación entre la aplicación Android y un dron de la marca, pero ésta tiene limitaciones ya que impide controlar de manera directa a varios drones al mismo tiempo.
La investigación del presente Trabajo se ha centrado en la creación del procedimiento a realizar para comunicar distintos terminales Android entre sí, desarrollar una aplicación que permita controlar y supervisar una misión realizada por varios drones desde un sólo cliente, y crear un elemento intermediario, que nos permita comunicar todos los terminales entre sí.

En la actualidad, no existe en el mercado ninguna aplicación, gratuita o de pago, que permita organizar a varios drones para la realización de las mismas tareas o de tareas complementarias. La aplicación desarrollada podría incorporarse a distintos sectores productivos en el presente, en cuanto permite la realización de tareas arduas o peligrosas, limitando sus costes tanto económicos como de riesgo humano, a la vez que minimiza alguna de las desventajas de estos nuevos vehículos, como puede ser la autonomía de vuelo.

A lo largo de este Trabajo Final de Grado se explican los procesos necesarios para implementar un cliente en Android, mediante el cual poder controlar a distintos drones de la marca. Entre algunos de los procesos adicionales a llevar a cabo, está la creación de un servidor en Node JS, el cual nos hará las funciones de servidor para poder almacenar y transferir información entre cada uno de los terminales.

Además, se hará uso del servicio de Google llamado Firebase, mediante el cual podremos realizar el envío de mensajes o notificaciones a cada uno de los terminales. Será a través de dichos mensajes, mediante los cuales podremos enviar de una manera indirecta notificaciones, al resto de terminales Android utilizados para llevar a cabo una misión.

1. Introducción

1.1. ¿Qué es un dron?

Según la Real Academia Española de la Lengua (RAE) un dron es toda aeronave no tripulada (UAV[1]). Un dron es todo aquel vehículo con la capacidad de volar, que se puede controlar de manera remota.

En cuanto a su diseño existen multitud de variantes, en formas, tamaños y características, dependiendo de la utilidad para la que se va a destinar.
Históricamente, los drones surgen con un diseño similar al de los aviones, debido a que originalmente se empezaron a utilizar con unos fines militares (I Guerra Mundial).

Ilustración 1. Dron Militar

Entre los diferentes fines para los que los ejércitos han venido utilizando estos vehículos podemos destacar los siguientes: reconocimiento de terreno, transporte de suministros a zonas de difícil acceso, defensa de tropas, etc.

A comienzos del siglo XXI, empresas como DJI (**D**à-**J**iāng **I**nnovations) intentaron introducir lo que hoy conocemos como drones no militares y de consumo. Dichos drones se pueden utilizar para actividades muy variadas, que pueden ir desde el ocio hasta la utilización para fines profesionales.

Ilustración 2. Dron de consumo

Ventajas de los UAVs

- Posibilidad de alcanzar zonas terrestres de difícil acceso para el ser humano.
- Es posible utilizar estos vehículos en zonas con un alto riesgo para las personas, ya sea por tratarse de una zona en guerra o propensa a sufrir catástrofes medioambientales.
- Permite la mejora del sector industrial al poder ser utilizados en distintos procesos productivos, lo que fomenta la capacitación e innovación en diversas áreas.
- Simplifica el proceso de tareas arduas y largas en el tiempo. Utilizar enjambres de estos vehículos aéreos permite reducir el tiempo necesario de una tarea.

Desventajas de los UAVs

- Son vehículos sensibles a los fenómenos meteorológicos como las tormentas, el viento o la actividad solar.
- Su capacidad de vuelo se encuentra bastante limitada por el tipo de combustible o energía utilizada. La mayoría de los UAVs usan baterías de Litio, a excepción de los de uso industrial o militar, que suelen utilizar combustibles como la gasolina o el keroseno.
- Tienen un radio de alcance limitado, ya que deben tener contacto con el emisor, para ir enviando y recibiendo instrucciones.
- Existe un retraso, que en algunos casos puede ser de segundos, en el envío y recepción de instrucciones por parte del personal en tierra.
- La utilización de este tipo de vehículos sin una serie de conocimientos mínimos o control del entorno donde se está usando, puede provocar problemas o dañar propiedades. Entre los daños o problemas más grandes que se pueden ocasionar, podemos destacar las situaciones

cada vez más frecuente, de usuarios que vuelan estos aparatos cerca de aeropuertos, zonas urbanas, edificios dotacionales, etc ..., con el alto riesgo que ello supone.

- Ofrecen la posibilidad de grabar o fotografiar objetos y personas, que para el caso de que se no se haya obtenido un consentimiento previo podría suponer una vulneración del derecho a la intimidad.

Además, en el ámbito de los drones, también podemos clasificar a las aeronaves en distintos grupos o categorías, dependiendo del nivel de capacidades que éstos disponen:

- Blanco: se utilizan para simular distintos tipos de ataques enemigos, en los sistemas de defensa de tierra y/o aire de los servicios de militares de un país.
- Reconocimiento: se trata de vehículos con una gran capacidad de vuelo, ya que pueden estar volando durante horas, días e incluso semanas. Este tipo de vehículos se utilizan, en la actualidad para propósitos militares.
- Combate: UAVs utilizados para librar batallas militares.
- Logística: vehículos aéreos diseñados para transportar cargas.
- Investigación y desarrollo: Estos vehículos se utilizan para probar e investigar sobre nuevos sistemas, que permitan a los UAVs realizar nuevas tareas, que faciliten la vida de las personas. Al mismo tiempo, permiten un uso sostenible, ya que desarrollan o pueden desarrollar funciones, con una menor emisión de gases de efecto invernadero.
  Es en este grupo de vehículos, en el que se centra este proyecto final de grado.
- UAV comercial: Son aparatos enfocados al ocio en general, que no pueden ser utilizados como juguetes, ya que acarrean una serie de riesgos. Se trata de vehículos, que se pueden controlar desde un mando en tierra, y además permiten la filmación y grabación de tomas aéreas.

Es necesario destacar que, dependiendo de la aplicación o finalidad del vehículo en cuestión, podremos encontrar un dron con una mayor o menor capacidad autónoma. En la parte alta de la tabla, encontraríamos los drones con una finalidad militar, mientras que en la parte inferior estarían los drones con fines comerciales.

1.2. Utilidades de los drones

Tal y como se ha indicado en el apartado anterior, los drones nos permiten realizar multitud de tareas de una manera mucho más sencilla y menos peligrosa. Pese a su precio en el mercado, cada vez son más las empresas que los adquieren para realizar distintos de trabajos aéreos, debido a que son herramientas muy útiles y versátiles. Así, cada vez es más común encontrar equipos profesionales, que los incluyen como una herramienta de trabajo más.

En cuanto a las posibles utilidades, que se le podría dar a un dron de consumo o profesional podemos encontrar las siguientes:

1. Filmación aérea: los UAVs permiten obtener filmaciones de una manera sencilla y con una serie de riesgos controlados. Así con unas pocas medidas de seguridad, se pueden llegar a obtener tomas de alta calidad, con un coste económico relativamente bajo.

   Un dron tiene un precio muy inferior a contratar un helicóptero, permisos de vuelo, operador… todo ello para obtener imágenes muy similares a las que se obtendrían con un dron. Resulta extremadamente práctico cuando se quiere filmar en zonas estrechas, a escasa altura…, zonas imposibles para un helicóptero. Es por este motivo, entre otros, por lo que cada día, más y más equipos de filmación profesional, están incluyendo drones como una herramienta esencial para su trabajo, puesto que les permite obtener tomas rápidas con una alta calidad, y sin la necesidad de la organización ni infraestructura que la utilización de un helicóptero supone.

Ilustración 3. Matrice 600

2. Entrega de productos de escaso peso: Empresas como Amazon están invirtiendo grandes cantidades de dinero, para poder entregar sus productos de una manera mucho más sencilla, eficaz y rentable. Ya que la energía necesaria para entregarle a un cliente un producto mediante un dron, resulta mucho menor, que la necesaria para una entrega con un vehículo de transporte terrestre, evita las aglomeraciones del tráfico, las normas de circulación terrestre, las rutas de reparto… en cambio esa otra forma de entrega consigue una mayor inmediatez, algo que en la actualidad es muy demandado, preferentemente, por el usuario de compras online.

Ilustración 4. Dron de Amazon

3. Situaciones de emergencia: este tipo de vehículos resultan extremadamente prácticos en situaciones de emergencia o altamente peligrosas, como puede ser un incendio forestal, terremotos, desastres naturales... o la desinfección de zonas contaminadas.

   Los UAVs podrían ayudar, enormemente, a la hora de contener o controlar un incendio forestal, por ejemplo, pueden volar a escasa distancia y obtener imágenes en tiempo real desde diversas posiciones o ángulos, informar de la temperatura de las distintas zonas, localización de personas o animales que se encuentren en peligro, comunicación directa con personas en riesgo para favorecer su salvamento…, todo ello, con un beneficio inmediato y directo como es la posibilidad de una organización más rápida y eficaz de los medios necesarios para la extinción del incendio.

   En cuanto a las tareas de desinfección, recientemente se han utilizado estos dispositivos en la lucha contra el virus COVID 19. En la provincia china de Wuhan se ha utilizado para la desinfección de calles, mercados, parques… de esta forma se ha evitado la exposición al virus de operarios de limpieza y desinfección, puesto que estos dispositivos pueden manejarse desde distancias kilométricas.

4. Cartografía: otro ámbito en el que se pueden utilizar este tipo de vehículos es en la cartografía o en el mapeado del terreno, tomando fotografías a distintas alturas que permitan a los expertos realizar análisis de las condiciones y características del terreno.

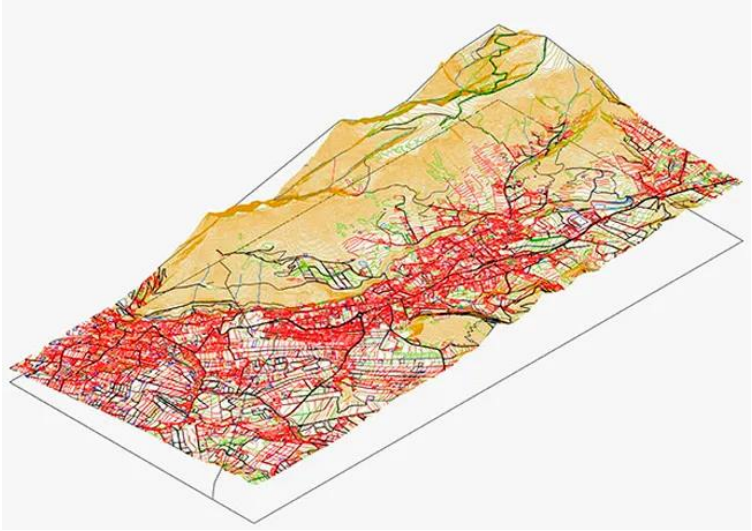


Ilustración 5. Cartografía de terreno

5. Rescates marítimos: entre el conjunto de utilidades para los que se pueden utilizar los drones, se ha podido comprobar que también resultan altamente útiles para obtener información desde otra perspectiva en rescates marítimos. Por ejemplo, cuando un barco se está hundiendo, tiene un incendio a bordo, tiene una fuga de crudo o una persona ha caído al agua.

Ilustración 6. Rescate marítimo

6. Rescates en lugares poco accesibles o de difícil acceso: tal y como se ha explicado en el punto anterior, existen ocasiones en los que los accesos a ciertas zonas resultan altamente complejos ya sea por vía aérea o terrestre. Es por ese motivo que un vehículo pequeño como un UAV puede resultar de gran utilidad, ya que permite hacer trabajos, que de otra manera resultan imposibles.

Ilustración 7. Rescate de difícil acceso

7. Vigilancia de fronteras: disponer de un vehículo aéreo de grandes dimensiones para controlar puntos fronterizos o la frontera de un país puede ser complejo y económicamente costoso. Es por ello, que disponer de una herramienta como es un dron puede ahorrar tiempo en desplazamientos y costes e incrementar la seguridad de las operaciones aéreas.

Ilustración 8. Zona fronteriza

8. Revisión de instalaciones en altura: una de las actividades en la que los drones avanzan a pasos agigantados, es en la revisión de trabajos en molinos eólicos e instalaciones de alta tensión. Debido a que los UAVs suelen disponer de cámaras de gran resolución y calidad de imagen por lo que pueden proporcionar una gran cantidad de información, sin riesgo humano.

Ilustración 9. Revisión línea alta tensión

9. Fines geológicos: esa es otra de las finalidades para las que pueden ser utilizados, por ejemplo, pueden resultar muy prácticos cuando se quieren tomar mediciones desde dentro o sobre el cráter de un volcán para determinar aspectos como el momento de su siguiente erupción…

Ilustración 10. Dron sobre volcán

10. Fumigación de campos de cultivo: el tratamiento de grandes extensiones de terrenos también podría realizarse con estos pequeños vehículos, ya que nuevamente el coste de ejecución de una operación de este tipo con un UAV resulta mucho más económica y seguramente más eficaz (puede volar muy bajo), que la que se haría con un avión fumigador.

Ilustración 11. Dron fumigando campo de cultivo

1.3. Aplicaciones existentes en el mercado

Las aplicaciones de vuelo para drones a través de dispositivos móviles resultan muy útiles y prácticas. Ofrecen información de alta importancia como: el porcentaje de batería, la calidad de señal recibida por el dron, número de satélites utilizados, distancia entre emisor y el vehículo, así como su localización, etc. Esta información se muestra al usuario de una manera clara e intuitiva.

La forma más básica de localización de un dron es a través de la información obtenida por su cámara, ya que, aunque lo perdamos de vista nos permite visualizar la zona en la que se encuentra para guiarlo tanto en su ruta como en la vuelta al receptor. Otra forma más técnica es a través del control de un icono que representa el dron y su seguimiento a través de la aplicación Google Maps. El usuario está perfectamente informado de la ubicación de su vehículo y, para el caso, que se produzca algún tipo de contingencia puede saber aproximadamente la localización del vehículo.

En cuanto a las aplicaciones disponibles en el mercado para controlar y/o ofrecer información relacionada con un dron en vuelo son muy pocas, ya que se trata de un producto que en la actualidad no está enfocado a la población en general, por lo que en su mayoría son no son gratuitas.

Algunas de las aplicaciones que se pueden encontrar en el mercado son las siguientes: DJI Ground Station Pro, Pix4d Capture, UGCS, Mission Planner, Litchi, DroneDeploy y DJI Go 4.

- DJI Ground Station Pro: es una de las últimas aplicaciones lanzada por la marca DJI para sus drones de uso profesional, como los de la serie Inspire o Matrice.
  Mediante dicha aplicación, que en la actualidad tan sólo se encuentra disponible para tabletas con sistema operativo iOS (dispositivos Apple), podemos crear una serie de misiones de una manera muy sencilla desde su interfaz de usuario y controlarlas en tiempo real.

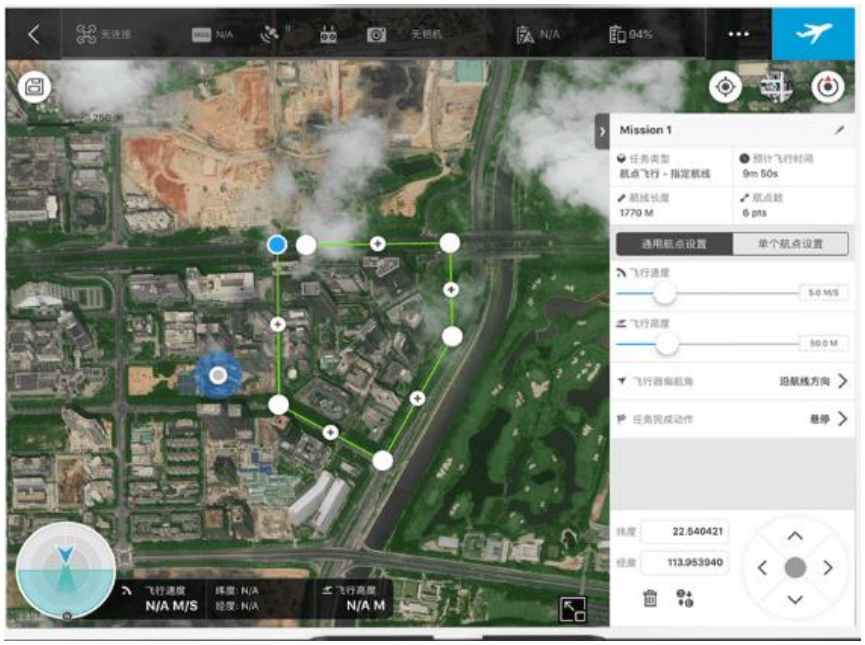


Ilustración 12. Interfaz de la DJI Ground Station Pro

Además, la aplicación permite la opción de guardar todos los datos generados en la misión en su propia nube. De esta manera, cualquier usuario de la aplicación puede estar seguro de que sus datos siempre se encontrarán disponibles cuando los necesite.

Otra posibilidad que ofrece esta aplicación es la de crear distintos tipos de usuarios. Se puede determinar qué información podrá ver cada usuario, algo que puede resultar muy útil en proyectos en los que se trabaje con un grupo amplio de personas.

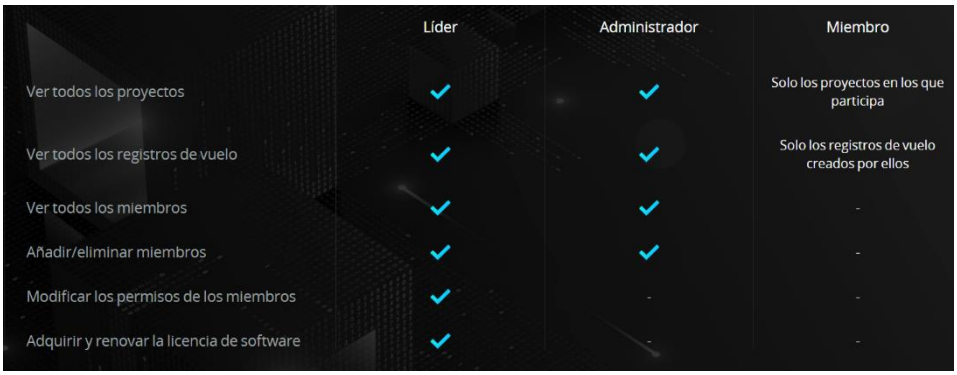


Ilustración 13. Tipos de usuario de DJI Ground Station Pro

Una de las utilidades más interesantes, que ofrece esta aplicación de entorno profesional para drones DJI, es la de mapear una zona de terreno que se selecciona previamente en el mapa de la aplicación. De esta manera calculará automáticamente la posición de cada uno de los Waypoints necesarios para cubrir por completo la zona marcada, y mapearla de forma óptima.

Además, esta aplicación permite modificar determinados parámetros, como la velocidad de vuelo durante la misión, la ratio de solapamiento frontal, la ratio de solapamiento lateral o el ángulo de vuelo.

Por último, hay que destacar que esta aplicación móvil también permite crear mapas detallados de estructuras de grandes dimensiones como la que se muestra en la siguiente imagen:

Ilustración 14. Mapa de Figura

- Pix4d Capture: es una de las aplicaciones más utilizadas en la actualidad por la comunidad de propietarios de drones de la marca DJI. Esta aplicación a diferencia de la anterior se encuentra disponible para dispositivos Android e iOS.

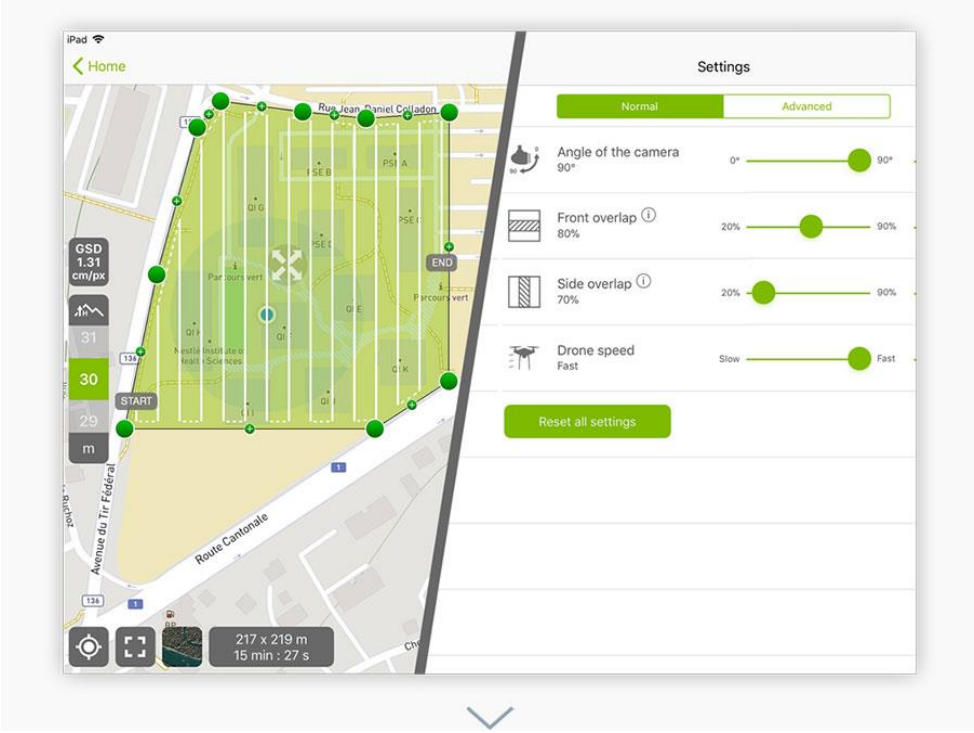


Ilustración 15. Mapa de Pix4d Capture

Al igual que sucede con la aplicación anterior se encuentra centrada en drones de la gama profesional y no de consumo.

En opinión de sus usuarios su funcionamiento resulta muy intuitivo puesto que, basta seleccionar una zona sobre un mapa para que automáticamente inicie su mapeo. Los tipos de formas que podemos dibujar sobre la pantalla tenemos los siguientes: cuadrado, rectángulo, doble cuadrado (cuadrados perpendiculares) y círculos.

A continuación, la aplicación nos permitirá introducir algunos aspectos para definir nuestra misión entre los que podemos destacar: la velocidad de vuelo, ángulo de inclinación de la cámara o el porcentaje de solape entre las distintas imágenes que capte el dron.

- UGCS: Es una aplicación cuya principal característica diferenciadora respecto a las dos anteriores es que no funciona con el "DJI Mobile SDK", no es para dispositivos móviles ni tabletas, sino únicamente está diseñada para funcionar en ordenadores o portátiles.

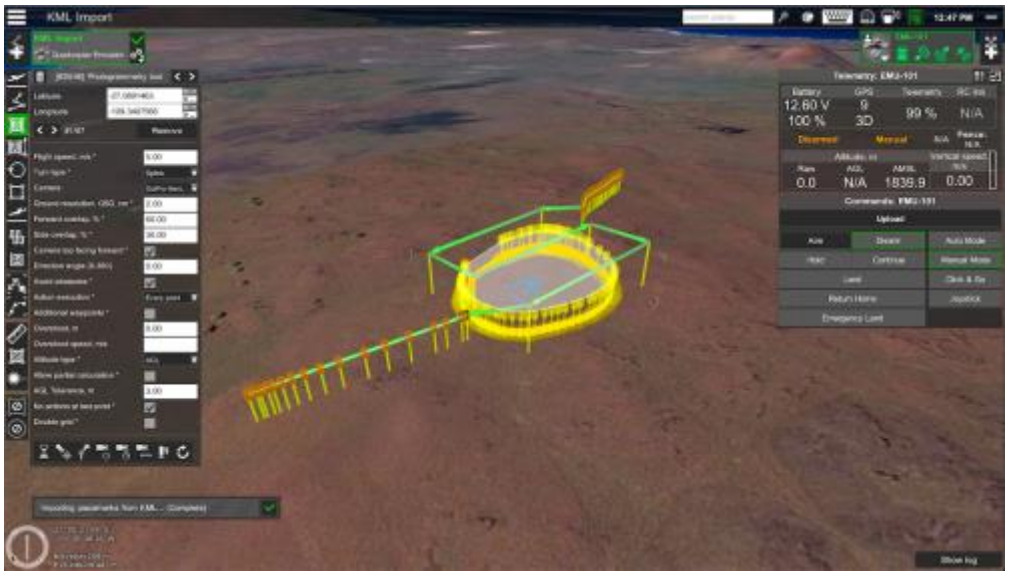


Ilustración 16. Captura de UGCS

Esta aplicación ofrece mucha más información durante la misión. Al igual que las dos aplicaciones anteriores se encuentra enfocada al mercado profesional, exigiendo a sus usuarios unos conocimientos avanzados para su manejo.

- Mission Planner: se trata de una aplicación también de uso profesional, que se ejecuta sobre el "Windows SDK". Es una aplicación que necesita un ordenador con Windows 8.1 o Windows 10, si bien existe una variante llamada APM Planner, para el sistema operativo Linux, que permite realizar casi las mismas funciones.

Ilustración 17. Captura de Mission Planner

Quizá la característica más destacable de esta aplicación es que es capaz de realizar una reconstrucción 3D, de una figura a partir de una serie de fotografías tomadas durante la misión.

También nos ofrece la posibilidad de realizar vuelos autónomos en círculos si lo deseamos, para ello simplemente deberemos incluir el radio del círculo y los Waypoints sobre los que tendrá que realizar las acciones, que el usuario determine oportunas.

Por último, hay que destacar que a diferencia de las anteriores aplicaciones ésta nos permite ajustar y cambiar una serie de parámetros de la controladora del dron para realizar diferentes configuraciones de ésta.

- Litchi: Se trata de una de las aplicaciones que más se utiliza a nivel consumo, seguida de la DJI Go 4. Litchi comparte propósitos con la aplicación que se va a desarrollar para presente Trabajo Final de Grado.

  Litchi es una aplicación que se encuentra disponible para dispositivos móviles Android e iOS. Además, es compatible con la amplia mayoría de drones de DJI, incluidos algunos de la gama profesional como pueden ser los Phantom 4 o la serie Inspire.

Ilustración 18. Captura de Litchi

Debemos tener en cuenta, que al igual que las otras aplicaciones ya comentadas se trata de una aplicación de pago. Aun así, muchos usuarios de drones DJI la recomiendan, debido a que ofrece posibilidades y opciones que la aplicación de referencia (DJI Go 4) de la marca no incluye, como pueden ser algunas funciones con los Waypoints.

En cambio, sí ofrece la posibilidad de realizar misiones con Waypoints incluso a drones como el DJI Spark que de manera oficial no lo soportan.

- DroneDeploy: se trata de otro planificador de vuelo para dispositivos móviles Android y iOS, compatible con la mayoría de los drones de la marca DJI. Se trata de una aplicación muy sencilla y de utilización intuitiva, muy similar a la aplicación Pix4D.

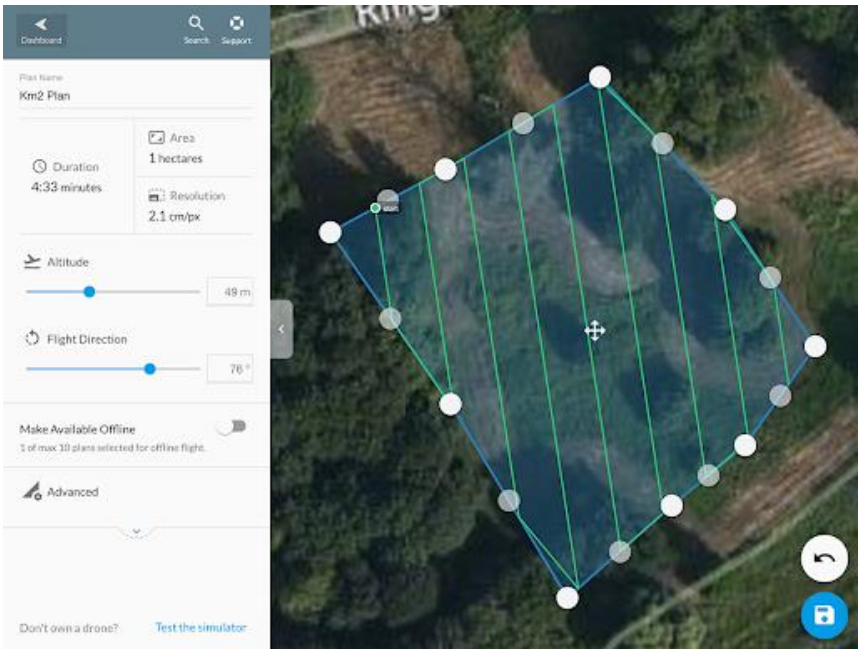


Ilustración 19. Captura de DroneDeploy

Al igual que sucedía con la Pix4D, para su utilización deberemos seleccionar la zona de terreno que queremos mapear, introducir la altura y dirección de vuelo y el porcentaje de SideLap y FrontLap. Una vez definidos todos los parámetros anteriores el dron comenzará a realizar la misión establecida y aterrizará de manera automática cuando la haya terminado.

- DJI Go 4: Se trata de la aplicación de referencia, ya que es la creada por la propia DJI para utilizarla con sus drones. Es una aplicación muy completa que no pone límites a la imaginación, ya que nos permite obtener un sin fin de información y aplicar gran número de configuraciones que transforman al dron en un vehículo muy versátil.

  Una de las características destacables de la aplicación es que ha conseguido automatizar las operaciones de despegue y aterrizaje del vehículo realizando el control de éstas con un sólo botón. Además, es capaz de transmitir vídeo en tiempo real en calidad HD.

  Dispone de modos más avanzados como el "TapFly" o el "Follow me". El primer modo permite indicar un punto en el mapa de tal manera que el dron se dirija al mismo de forma automática. Y con el modo "Follow me" el dron puede seguir automáticamente a un objeto o persona en movimiento.

Por último, la aplicación también incluye la posibilidad de introducir varios Waypoints a los que el dron volará de manera automática, de acuerdo con el orden introducido por el usuario.

Ilustración 20. Captura de DJI Go 4

El problema que desde mi punto de vista tiene esta aplicación, es que DJI, fundamentalmente, dota de las mismas características a su aplicación para sus drones de consumo que para los de su gama profesional, sin tener en cuenta que, en realidad se trata de vehículos con un perfil de usuario con diferentes necesidades.

Conscientes de ello, DJI sacó de manera oficial y pública a todo el mundo su SDK en distintas variantes como son: Mobile SDK, Windows SDK, Payload SDK y Onboard SDK, de esta manera proporcionó a los usuarios de todo el mundo la posibilidad de construir aplicaciones enfocadas a sus actividades específicas, lo que ha sido un gran acierto ya que como hemos visto existen aplicaciones de distinto tipo y para distintas plataformas.

1.4. Ventajas y utilidades de trabajar con un enjambre de drones

Como he dicho anteriormente el uso profesional de los drones se ha extendido de tal forma que las empresas lo vienen utilizando como un instrumento fundamental en su equipo. Y ahora, en lugar de asignarles misiones individuales, se les programa para misiones que supongan la utilización de varios drones a modo de equipo con tareas complementarias. Con ello, se subsana una de sus principales desventajas, su reducida autonomía, que puede llegar a ser un inconveniente en muchas ocasiones. Así el uso de varios drones al mismo tiempo se está convirtiendo en algo cada vez más frecuente.

Otro aspecto, que debemos tener en consideración es que los drones resultan muy prácticos para tareas como la fumigación de campos, utilización especificada en el punto 1.2 de este Trabajo Final de Grado. Estos vehículos pueden volar a una muy baja altura, lo que mejora la eficacia de los productos que se utilizan y el riesgo tóxico para zonas habitadas.

Los drones resultan increíbles para arduas tareas, pero son poco prácticos cuando la misión se refiere a una zona muy extensa, como por ejemplo fumigar un territorio, fundamentalmente por un problema de autonomía de sus baterías, pero ahí es donde entran en juego los denominados enjambres.

Un enjambre es un conjunto de drones, que realizan una misma tarea en conjunto, es decir, si tenemos que fumigar el mismo campo que en el párrafo anterior, pero en lugar de tener un dron tenemos ahora cincuenta, la tarea la podremos realizar de una manera mucho más eficiente, debido a que todo el producto químico se podría aplicar sobre las plantas en el mismo día, ya que cada dron tan sólo tendría que hacer una cincuentava parte del campo.

Por último, señalar que, ya se han podido ver a enjambres de drones realizar distintas misiones complementarias, como se pudo ver en el cielo de Shanghái al inicio de este 2020, cuando se utilizaron más de 2.000 drones iluminados para conseguir el efecto de fuegos artificiales teledirigidos, si bien no se trata de una actividad productiva, sí es gráfica a los efectos de visualizar las muy variadas aplicaciones y combinaciones que se pueden realizar con el uso de muchos drones.

Ilustración 21. Nuevo año 2020

1.5. Cómo mejorarían las aplicaciones utilizando un enjambre de drones

De acuerdo con lo explicado en el punto anterior, procedemos a analizar cómo las aplicaciones expuestas en el apartado 1.3 podrían verse mejoradas si se pudiese utilizar este sistema.

- DJI Ground Station Pro: para el caso particular de esta aplicación el poder utilizar un enjambre de drones la haría mucho más útil y ventajosa para muchas empresas. Un sólo dron no tendría que recorrer todos los Waypoints establecidos por la aplicación, sino que se repartirían de la forma más equitativa las tareas. Esa funcionalidad podría ser muy útil cuando para completar una misión fuera necesaria más de una batería, puesto que al dividir la misión principal en misiones más pequeñas éstas se completarían antes de ver como la batería se ha vaciado por completo.

- Pix4d Capture: esta aplicación tendría una importante mejora similar a la que se ha mencionado en el caso anterior. Esto se debe a que, si deseamos capturar imágenes de una extensión de terreno y, ésta es muy extensa, deberemos tener en cuenta la autonomía de nuestro dron, y si seleccionamos una zona muy grande, el dron puede tener que abortar la misión antes de completarla, para regresar a su lugar de despegue por falta de batería. Entonces al igual que sucede con la DJI Ground Station Pro se podría dividir la sección a

realizar entre el número de drones disponibles con la finalidad de maximizar el tiempo sobre el terreno.

- UGCS: cabe destacar que a diferencia de las anteriores aplicaciones ésta sí contempla que nos podamos quedar sin batería durante la misión, es decir, la aplicación soporta misiones en las que se tengan realizar varios vuelos y, por lo tanto, varios cambios de batería para completar la siguiente etapa de la misión. Por ello, la mejora de esta aplicación no debería ser muy compleja.

- Mission Planner: esta aplicación se debe ejecutar sobre Windows 8.1 o Windows 10, y al compartir bastantes aspectos con las anteriores, el añadir la funcionalidad de trabajar con enjambres la haría mucho más atractiva y práctica.

- Litchi: a diferencia de las anteriores y tal y como se ha expuesto en el punto 1.3, esta aplicación no está pensada como las anteriores para su uso profesional y, por lo tanto, no sería necesario el dividir la tarea, sino que la mejora se correspondería con la posibilidad de obtener vídeos y fotografías de una misma ruta, pero desde distintas perspectivas, es decir, fotos o vídeos realizados a distintas alturas y a distintas distancias de cada uno de los "Waypoints".

Ilustración 22. Enjambre de drones

1.6. Objetivos del proyecto

El principal objetivo de este Trabajo de Final de Grado consiste en desarrollar una aplicación que funcione en dispositivos móviles y tabletas con cualquier versión del sistema operativo Android (Versión 4.4 o superior) mediante la cual podamos automatizar ciertas tareas con múltiples UAVs.

En caso de disponer de una versión anterior dicha aplicación no funcionará, ya que la mínima versión que admite esta aplicación para su correcto funcionamiento es la API de Android número 19.

Esta aplicación debe realizar funcionalidades similares a las expuestas en el punto 1.3. En particular debe permitir tareas como las que se pueden realizar con la aplicación Litchi, es decir, debe ofrecer la posibilidad de comunicarse con un dron de la marca DJI para indicarle una serie de tareas a realizar. Para el desarrollo de dicha aplicación ha sido necesario el uso del DJI Mobile SDK.

El DJI Mobile SDK no es más que una API especialmente diseñada por el fabricante para que desarrolladores de todo el mundo pudieran implementar sus propias aplicaciones y utilizarlas con sus propios drones. El uso de esta API es totalmente gratuito, y simplemente necesitaremos crearnos una cuenta como desarrollador para poder implementar hasta un total de veinte aplicaciones con su plan gratuito.

Además, tal y como se ha expuesto en punto 1.5 de la presente memoria, el trabajo colaborativo de drones puede resultar altamente práctico así que, la aplicación debe ser capaz de poder trabajar con múltiples drones al mismo tiempo, algo que como se ha podido comprobar no se encuentra disponible en ninguna otra aplicación del mercado incluyendo las de pago.

Por ese motivo, uno de los objetivos principales de este Trabajo Final de Grado es el crear una aplicación para dispositivos Android, en la que indicando una serie de Waypoints, un dron de la marca DJI irá a cada una de las coordenadas GPS indicadas por el usuario y realizará alguna acción sobre cada una de ellas. Además, esa misma ruta podrá realizarse con varios drones al mismo tiempo, es decir, debe existir la posibilidad de que varios drones de la marca puedan ejecutar de manera simultánea la ruta introducida por el usuario, pero a una altura distinta y como una formación. Lo que se busca conseguir con estas distancias entre drones es obtener una mayor seguridad, de que sistemas como el GPS no van a tener un error y los drones vayan a chocar en el aire o que una racha de viento empuje los drones hasta colisionar en el aire, lo cual podría resultar fatal para los mismos dependiendo de la altura de vuelo que tengan en ese momento.

La ruta se trazaría con la información introducida por el usuario, destacando los siguientes parámetros de configuración: coordenadas GPS de los Waypoints, altura mínima de vuelo, altura entre drones, y la distancia que debe existir entre un dron y el Waypoint. De esta manera, se pretende que un usuario desde un móvil introduzca los Waypoints a los que quiere que los drones se dirijan, así como su orden y que cuando las coordenadas relativas a dicho punto GPS se encuentren calculadas para cada uno de los drones, el mismo usuario que ha indicado los Waypoints, pulsando un botón envíe la orden a todos los drones para que inicien la misión.

Debido a que el DJI Mobile SDK está limitado a que cada dron debe estar conectado a un mando y éste a su vez a un dispositivo móvil, debemos tener en cuenta que si vamos a volar cinco drones vamos a necesitar cinco aplicaciones móviles y, por lo tanto, tendremos que solventar la problemática de enviar la señal de inicio de misión a todos los dispositivos móviles al mismo tiempo para que la formación se pueda mantener en todo momento.

Por ello, el presente trabajo se centra en alcanzar los siguientes objetivos:

1. Aplicar conceptos adquiridos durante los distintos cursos del grado en Ingeniería Informática por la Universidad de Alicante. Por lo tanto, se busca realizar el desarrollo de una aplicación utilizando los conocimientos adquiridos en asignaturas como: Programación 3, Redes de Computadores, Interconexión de Redes, Desarrollo de Aplicaciones para Internet, Diseño de Bases de Datos, Administración de Sistemas Operativos en Redes de Computadores, Sistemas Distribuidos o Análisis y Especificación de Sistemas Software.

2. Aprender a utilizar herramientas como el Mobile SDK para automatizar múltiples tareas con UAVs, ya sea, utilizando un único dron o múltiples de ellos.

3. Aprender a desarrollar aplicaciones desde cero para dispositivos Android, incluyendo la implementación de la parte lógica, como de vistas gráficas utilizables para distintos dispositivos de distinto tamaño y resolución.

4. Conocer el uso y aplicación de SDKs como el de Google Maps para Android, con la intención de integrar dentro de aplicaciones propias los mapas de Google Maps.

5. Aplicar servicios modernos como de Firebase de Google, para poder notificar a usuarios de una aplicación desarrollada por uno mismo, de una manera sencilla, mediante el uso de sus propios servidores.

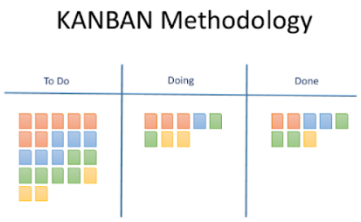
6. Aplicar los conocimientos adquiridos en el grado para crear una aplicación con la arquitectura cliente-servidor.

7. Conocer el uso de herramientas como el simulador de DJI, para poder comprobar el correcto funcionamiento de nuestras aplicaciones para drones de la marca. Para ello, será necesario el uso y conexión a drones reales.

8. Investigar el procedimiento o los posibles procedimientos para crear una aplicación, que permita controlar un enjambre de drones desde un único terminal Android.

9. Desarrollar utilizando los frameworks “express” y “knex” en un servidor en Node JS, para atender todas las peticiones HTTP de cada uno de los distintos clientes Android.

1.7. Metodología

Metodología	Kanban	
Motivos	• Imposible emplear el mismo tiempo todas las semanas • Falta de experiencia previa en los campos a tratar • Previsiones mal realizadas por la falta de experiencia	
Ventajas	• Se trata de una metodología ágil y flexible • Resulta sencillo, eficaz e intuitivo	

A lo largo del presente apartado se procederá a explicar la metodología utilizada para llevar a cabo el proyecto, así como las principales razones por las que se ha elegido dicha metodología.

Debido a la temática de este Trabajo Final de Grado, la creación de una aplicación para el manejo de múltiples drones desde un mismo cliente Android, y no disponiendo de ningún tipo de experiencia previa en este campo, resultaba fundamental utilizar una metodología que fuera ágil, flexible y sobre todo sencilla, que permitiera ir modificando las estimaciones realizadas en un inicio para cada una de las tareas, que componen el proyecto.
La falta de experiencia en el campo de las aplicaciones para drones ha supuesto que algunas de las tareas han llevado un tiempo muy superior al previsto, y ello, dificulta la estimación del coste de

realización de cada una de las partes que componen el proyecto. Esa falta de experiencia previa unida al trabajo propio de seguir el cuatrimestre (asignaturas, entregas de trabajos, prácticas externas) iba a imposibilitar emplear todas las semanas el mismo número de horas.
En el caso del servidor, ya se había realizado un programa similar en la asignatura Desarrollo de Aplicaciones en Internet (DAI), por lo que, no iba a resultar muy complejo este procedimiento, al disponer de experiencia previa en la realización de un servidor en NodeJS.

Por lo que tras una primera investigación me decidí por la metodología "Kanban", que está ganando popularidad en distintas empresas de todo el mundo, debido a la gran sencillez con la que se puede generar un flujo eficaz y eficiente de trabajo entre los miembros de un equipo.

Kanban es una metodología que utiliza tarjetas para indicar a través de una serie de etapas las cosas pendientes de hacer, las tareas que se encuentran en proceso por alguno de los miembros del equipo y las que ya se encuentran terminadas o completadas.

Ilustración 23. Kanban

Esta metodología puede resultar muy beneficiosa para un grupo de desarrolladores, ya que se puede ver quien está realizando o ha realizado cada tarea. Por último, también resulta altamente recomendable incluir en cada una de las etiquetas la duración prevista en horas, que deberían ser necesarias para completar dicha tarea. Tal y como se ha mencionado, este último aspecto en esta ocasión no ha resultado muy útil por la falta de experiencia previa en el cambio de los drones.
Entre algunas de las ventajas de Kanban podemos destacar las siguientes:

1. Organización y Colaboración: permite ver de una manera muy visual y rápida, gracias a las distintas columnas aspectos como el flujo de trabajo actual o el número de horas pendientes previstas hasta finalizar el proyecto.

2. Distribución del trabajo: permite la división de la carga de trabajo entre los distintos miembros de una manera sencilla y eficaz, aunque como se ha indicado previamente, ésta podrá modificarse en cualquier momento debido a distintos factores como puede ser la mala previsión en el cálculo de horas.

3. Rendimiento: gracias al movimiento de cada una de las etiquetas, Kanban permite a los miembros de un equipo de desarrollo comprobar en cualquier momento su rendimiento hasta la fecha, es decir, permite verificar si el proyecto va de acuerdo a la planificación realizada, va con retraso o va avanzando más rápido de lo previsto.

Por último, destacar que para llevar a cabo el presente Trabajo de Final de Grado se podrían haber utilizado herramientas como Trello o Github Project entre otras muchas online y gratuitas existentes en la actualidad.

Para el caso del presente proyecto se decidió utilizar la herramienta incluida dentro de cualquier repositorio de Github, que permite ir moviendo las distintas etiquetas a lo largo de las columnas tal y como se puede ver en las imágenes inferiores:

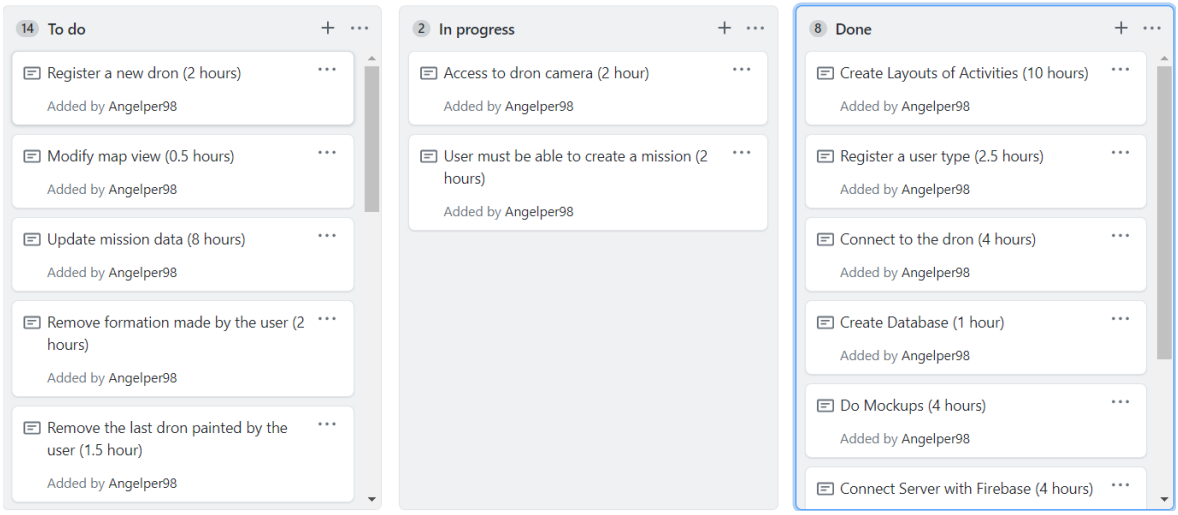


Ilustración 24. Kanban Github 1

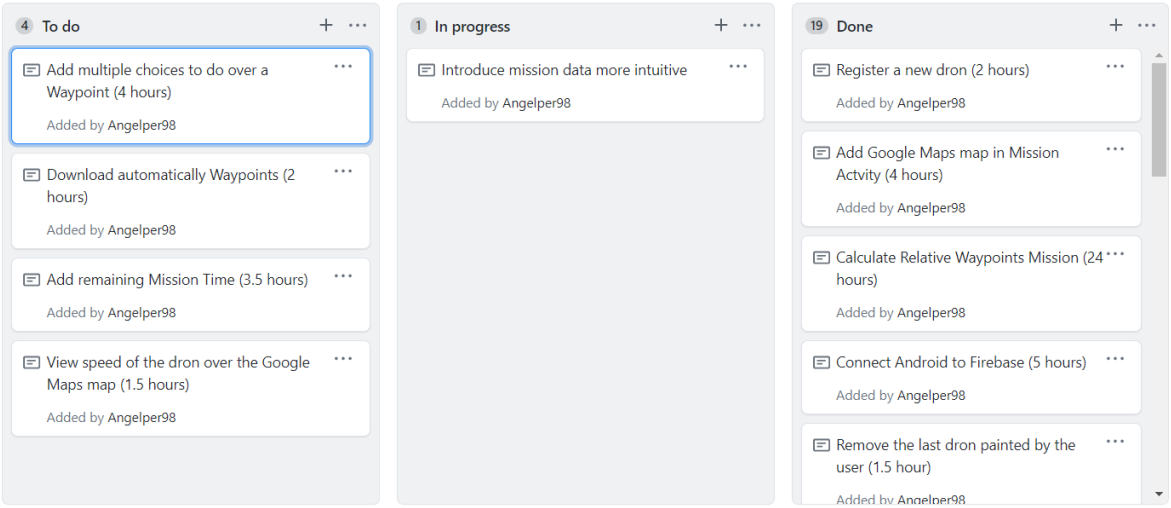


Ilustración 25. Kanban Github 2

En cuanto al coste de la aplicación desarrollada tenemos el siguiente:

- Aprender el funcionamiento de Android SDK -> 24 horas
- Entender estructura y funcionamiento de proyecto Gradle -> 8 horas
- Estudiar documentación Mobile SDK -> 30 horas
- Hacer las guías del Mobile SDK -> 75 horas
- Hacer todos los mockups y pensar la utilidad de cada componente de la vista -> 4 horas
- Crear vistas que se adapten bien a móviles y tabletas de distintos tamaños -> 10 horas
- Registrar en una misión a un tipo de usuario -> 2.5 horas
- Crear nuevas misiones -> 2 horas
- Acceder a una misión -> 2.5 horas
- Registrar un nuevo dron en una misión determinada -> 2 horas
- Conectar la aplicación con un dron DJI -> 4 horas
- Acceder a la cámara del dron -> 2 hora
- Crear base de datos y proyecto del Servidor -> 1 hora
- Cargar mapa de Google Maps -> 4.5 horas
- Poder modificar el tipo de mapa a mostrar -> 1.5 hora
- Actualizar los datos de una misión -> 8 horas
- Usuario puede dibujar la formación -> 3.5 horas
- Usuario puede eliminar todos los drones dibujados -> 2 horas
- Usuario puede eliminar el último dron dibujado -> 1.5 horas
- Calcular los puntos relativos para los drones -> 24 horas
- Descargar y dibujar los Waypoints en el Mapa -> 6 horas
- Conectar los clientes con Firebase -> 5 horas

- Mandar mensajes desde el servidor con Firebase -> 6 horas
- Iniciar, Pausar y Detener misiones -> 8 horas

Por lo tanto, el coste de desarrollo de la aplicación para este Trabajo Final de Grado ha supuesto un total de 237 horas, si valoramos la hora a 12 euros, la aplicación tiene un coste total de aproximadamente 2.844 euros.

Finalmente, destacar que, además de disponer de un repositorio en Github para disponer de todo el código fuente y del control del proyecto, también se han utilizado otras herramientas como: Android Studio, Visual Studio Code, Virtualbox o Microsoft Visio.

2. Desarrollo del Trabajo

2.1. Posibles aplicaciones del proyecto

En relación a las posibles aplicaciones prácticas que se le pueden dar a este tipo de herramientas, ya sea en un entorno profesional como en un ámbito más enfocado a la afición, se ha podido comprobar que las posibilidades son prácticamente infinitas debido a la alta capacidad que tienen este tipo de vehículos para poder adaptarse a cualquier tarea.

Además, debemos tener en cuenta que debido a que esta aplicación sí que permite el uso de múltiples aeronaves al mismo tiempo, siempre existirá la posibilidad de realizar la misma misión, pero con los drones en distinta formación, distancia o altura. Todo esto combinado podría proporcionarle al usuario una gran herramienta de cara a su actividad profesional.

Entre algunas de las posibilidades que podría proporcionarnos esta aplicación podemos destacar las siguientes:

- Fotografía y Vídeo: al existir la posibilidad de colocar los drones en distintas posiciones relativas al "Waypoint", el cual habremos marcado en la pantalla del dispositivo móvil previamente, podremos obtener distintas perspectivas ya sea en ángulo y altura del objetivo sobre el que deseamos generar material de filmación.

- Extinción de incendios: otra gran utilidad enfocada al entorno profesional puede ser el control y extinción de incendios. Los drones controlados por la aplicación podrían servir para transmitir una gran cantidad de información y de excelente calidad a los profesionales al cargo de la extinción del incendio. Dichos profesionales podrían apreciar desde distintos ángulos como se está llevando a cabo las tareas por los bomberos, así como si existe algún punto por el que el incendio se está extendiendo con rapidez.

- Control de plagas: otra circunstancia en la que pueden ser muy útiles los drones en enjambres es a la hora de aplicar sustancias nocivas para ciertos insectos. Tal y como se ha indicado en apartados previos, estos vehículos pueden volar a baja altura lo que les permite aplicar el producto de una manera mucho más directa. Además, al poder indicar una formación por parte del usuario se puede fumigar de manera óptima y eficiente.

- Control de fronteras físicas: también pueden resultar muy prácticos para el control y supervisión de fronteras, ya que se puede tener a drones volando sobre dichas zonas proporcionando información desde distintas perspectivas, lo que para los agentes puede resultar muy práctico.

- Control de infraestructuras: estos pequeños vehículos aéreos también pueden ser utilizados para el control de grandes infraestructuras civiles como pueden ser embalses, puentes o centrales eólicas, ya que resultan extremadamente prácticos debido a que pueden ofrecer una perspectiva totalmente diferente. Dicha perspectiva es sumamente sencilla en comparación a lo que costaría obtener esa información por parte de un ser humano sin esta herramienta.

- Control de playas y zonas costeras: esta aplicación también puede resultar muy práctica para el rescate de bañistas que se van mar adentro y luego no pueden volver a la playa por alguna circunstancia. La aplicación está diseñada para trabajar con un grupo amplio de drones, los cuales pueden ir desde 1 hasta 10 (se podría aumentar el número de manera sencilla) y así lanzar el número necesarios de vehículos para llevar a los bañistas de una manera rápida y eficaz salvavidas a los que se puedan agarrar mientras los servicios de rescate llegan.

- Información sobre el tráfico en una ciudad: la aplicación también podría utilizarse para que los drones transmitieran información del estado del tráfico en ciudades grandes como pueden ser Madrid o Barcelona.

- Control de eventos de gran afluencia: los vehículos aéreos como los drones controlados en forma de enjambre, con una aplicación como la de este Trabajo de Final de Grado podrían resultar muy prácticos a la hora de controlar desde el aire eventos masivos de personas como pueden ser eventos deportivos a nivel mundial (Olimpiadas) o festivales de música, así como pruebas deportivas en núcleos urbanos, maratones, pruebas ciclistas....

2.2. Funcionamiento del SDK de DJI

Tal y como se ha indicado en apartados previos, para poder llevar a cabo el desarrollo del principal objetivo del presente proyecto ha sido necesario la inclusión del DJI Mobile SDK en la aplicación cliente para nuestro dispositivo Android. Así será el propio SDK el que nos permita conectarnos a los drones de la marca y poder realizar las acciones que queramos con ellos.

DJI Mobile SDK no es más que un software que la marca de drones DJI ha diseñado para que, desarrolladores de todo el mundo puedan disponer de una herramienta para acceder a todas las capacidades y utilidades de sus drones. Gracias al uso de esta herramienta podemos ver como el proceso de desarrollo se reduce enormemente, ya que, prácticamente, cualquier componente o función necesaria para un profesional se encuentra correctamente desarrollada y comprobado su funcionamiento por la marca.

Además, al utilizar esta herramienta gratuita podremos obtener un gran nivel de abstracción, debido a que como se ha indicado previamente existen multitud de utilidades ya creadas y, por lo tanto, no tendremos que preocuparnos de implementar todas y cada una, de las funciones que incluya nuestra aplicación. Las utilidades que actualmente podemos añadir a nuestra aplicación son: estabilización en vuelo, control de la batería o la calidad de la señal.

El Mobile SDK incluye una librería/framework que puede ser importada a un proyecto de Android Studio, para proporcionar acceso desde cualquier clase o "Activity" que creemos, en nuestro proyecto al dron DJI. Entre sus características más relevantes destaca el simulador, que es una herramienta fundamental para desarrollar cualquier aplicación, que vaya a conectarse a drones DJI; estos vehículos no son baratos y es fundamental comprobar el correcto funcionamiento de nuestra aplicación en un entorno controlado como puede ser un simulador, donde no corramos peligro de romper nuestro UAV, cosa que sí podría ocurrir si probamos nuestra aplicación directamente en el mundo real.

El simulador es altamente preciso y fiable ya que la simulación no se lleva a cabo utilizando la CPU del ordenador donde esté instalado el simulador, ni tampoco sobre la del móvil Android utilizado, sino sobre la propia CPU que cualquier dron de la marca lleva en él. De esta manera lo que se consigue es obtener la certeza de que, si algo funciona en el simulador, funcionará en la vida real posteriormente, ya que va a ser ejecutado también por el propio UAV.

Durante la fase de testeo podremos comprobar una gran cantidad de valores útiles como pueden ser: la velocidad sobre los ejes X, Y y Z o las coordenadas GPS donde se encuentra el dron. Todos estos valores y muchos más los podemos ver en el recuadro de color verde, que aparece en la siguiente imagen:

Ilustración 26. Simulador DJI Assistant

Así para poder desarrollar cualquier aplicación ya sea para dispositivos móviles iOS o Android debemos disponer de un dron de la marca, de otro modo sería imposible. Por último, destacar que, para el desarrollo del presente proyecto en mi caso he utilizado un DJI Mavic Air, mediante el cual he podido ir probando todas y cada una de las funcionalidades relacionadas con el SDK.

Otra gran utilidad que se incluye dentro de esta librería es lo que DJI llama como UX. Este paquete nos permitirá de una manera muy rápida y sencilla integrar características, que la aplicación de referencia DJI Go 4 incluye. Alguna de estas características puede ser: porcentaje restante de batería, despegue y aterrizaje automático o la posibilidad de hacer videos o fotos. En resumen, mediante este paquete podremos disponer prácticamente de una aplicación como la DJI Go 4.

Ilustración 27. DJI Go 4

Para poder desarrollar cualquier aplicación con el Mobile SDK necesitaremos de un ordenador con sistema operativo Windows 8.1, Windows 10 o Linux, y así poder programar la aplicación utilizando Android Studio. Además, como es obvio, debemos disponer de un móvil Android físico ya que tal y como se informa y he podido comprobar, las aplicaciones que incluyen este framework tan sólo funcionan en dispositivos físicos. Y para poder correr la simulación deberemos ejecutarlo sobre un ordenador con Windows, debido a que el simulador no está disponible para Linux.

Una vez conocidas las herramientas necesarias para desarrollar con el Mobile SDK, deberemos conocer cómo funcionan para poder integrarlas en nuestra aplicación Android.

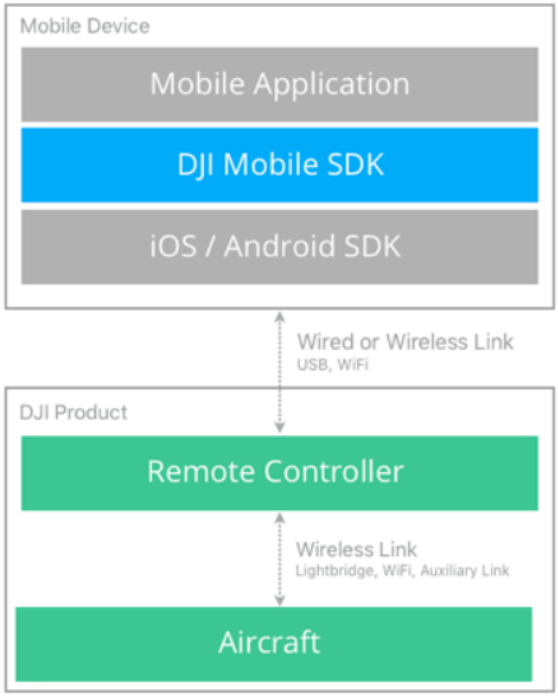


Ilustración 28. Proceso de comunicación SDK con UAV

Tal y como se puede apreciar en la imagen anterior (obtenida de la documentación oficial) el SDK nos hace de intermediario para comunicarnos con el dron. En cuanto al proceso de comunicación entre el móvil Android y el dron es el siguiente:

1. La aplicación utiliza el Mobile SDK para enviar alguna instrucción
2. El Android SDK coge dicha instrucción y la transfiere al control remoto del dron, mediante un cable USB conectado entre ambos o vía Wi-Fi al punto de acceso creado por el mando.
3. El control remoto recibe la instrucción y la transmite vía Wi-Fi a UAV.

A modo de aclaración decir, que debido a que la arquitectura de la aplicación desarrollada para este proyecto ha sido la cliente-servidor, será estrictamente necesario conectar el control remoto y el terminal Android mediante USB. Esto es debido a que el punto de acceso que genera el control

remoto no tiene salida a internet y, por lo tanto, no habría posibilidad de enviar y/o recibir información del servidor, algo fundamental para el correcto funcionamiento de la aplicación cliente.

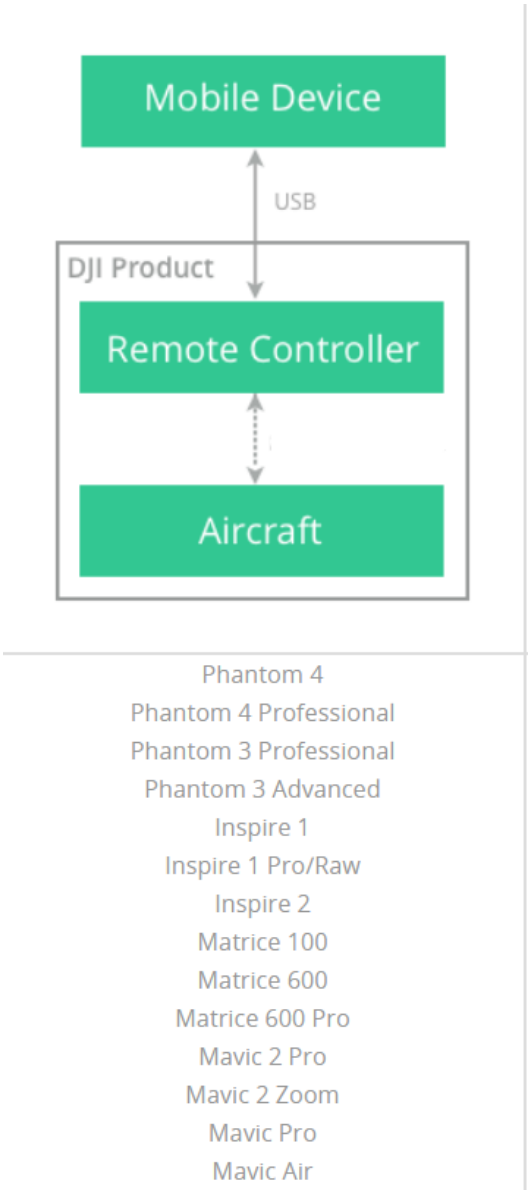


Ilustración 29. Proceso de comunicación Móvil con UAV

De esta manera, tal y como se puede ver en la imagen, para el desarrollo de la aplicación vamos a usar el proceso de comunicación.

Además, se trata de una de las tres posibles maneras, que el SDK permite, pero es ésta la opción seleccionada por ser la que tiene una mayor compatibilidad con los distintos drones de la marca. Por lo tanto, al utilizarla será posible emplear un abanico mayor de drones.

Se ha podido comprobar que la arquitectura con la que se ha desarrollado y funciona el Mobile SDK es altamente extensible. Además, todas las clases de un producto y sus componentes son abstractas, y mediante ellas podemos controlar los distintos productos de la marca DJI con el mismo código, es decir, no será necesario crear distintas variantes de una misma aplicación en función del dron que se vaya a emplear.

Algo que sí debemos valorar, es que no todos los drones de la marca tienen disponibles las mismas funciones, como es lógico los drones de la gama profesional tendrán características y funciones, que drones de la gama de consumo no dispondrán. Todo esto, ya se encuentra contemplado por la propia marca, ya que según informa en su documentación, si utilizamos alguna funcionalidad que no se encuentre soportada por un UAV en particular, simplemente no funcionará, pero en ningún caso saltará una excepción o error causado por dicha ejecución.

Por ello, de acuerdo con lo indicado, los desarrolladores no deben preocuparse por dar soporte a los nuevos productos de la marca que van saliendo al mercado, ya que éstos funcionarán correctamente con la aplicación desde el principio.

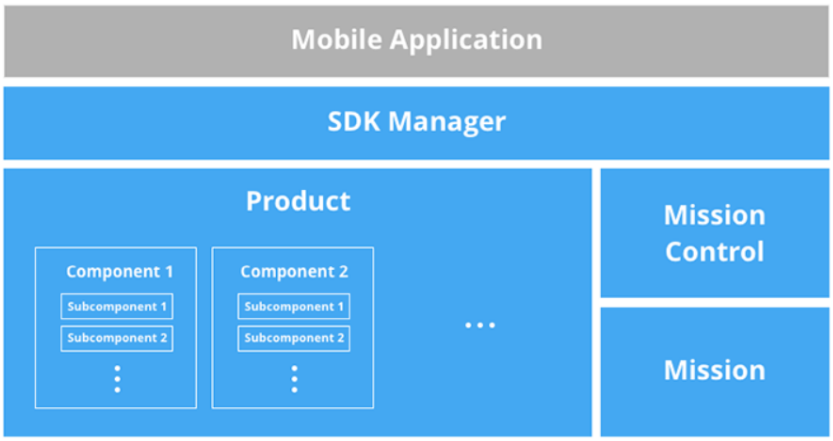


Ilustración 30. Diseño Mobile SDK

El SDK Manager se encargará de registrar el SDK para validar la aplicación en los servidores de DJI. Este paso es fundamental, ya que, si no se registra la aplicación correctamente, no se podrán utilizar las funcionalidades del "framework".

La clase Producto es la que representa, en este caso, a los drones de la marca. Además, la clase contiene las propiedades básicas y los componentes principales del producto.

Dentro de la clase Producto es donde encontramos la clase Componente, que representa a distintos componentes del dron como pueden ser el Flight Controller, que es básico para realizar cualquier operación, ya que se trata del elemento que supervisa y controla el vuelo de la aeronave.

La clase Misión es la que nos permite describir misiones con Waypoints, que es el tipo de misión empleada para llevar a cabo este proyecto, ya que si no existieran este tipo de misiones no habría sido posible implementar la aplicación.

Por último, en cuanto al Control de la Misión debemos tener en cuenta que se trata de la clase que supervisa el correcto desarrollo de la misión, así como, determina la serie de acciones que tiene que llevar a cabo el dron.

2.3. Planificación de la aplicación para Android

2.3.1. Manuales empleados de DJI

Para el desarrollo de esta aplicación de Trabajo de Final de Grado ha sido fundamental el uso de la documentación del DJI Mobile SDK ya que, al tratarse de una programación tan específica para unos productos con un mercado limitado, no existen muchos sitios web en los que se ofrezcan soluciones a posibles problemas. Es decir, es fundamental entender el correcto funcionamiento del SDK, cómo integrarlo y utilizarlo en nuestras aplicaciones antes de empezar.

Por ello, la propia marca DJI incluye en su documentación una serie de ejemplos básicos que nos servirán de base para crear prácticamente cualquier aplicación que nos propongamos. Además, si algo no nos ha quedado claro, DJI nos pone a nuestra disposición su repositorio de Github en el que podremos acceder al código completo del tutorial. Algunos de estos tutoriales han sido fundamentales para poder desarrollar la aplicación, ya que explican de manera detallada cómo implementar funcionalidades básicas.

Algunas de las guías que más útiles han sido, son los siguientes:

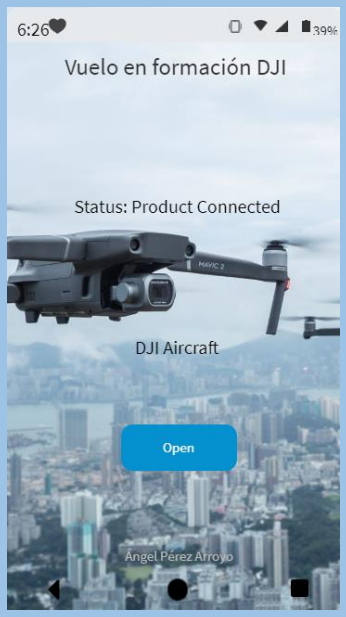
1. Cómo integrar el SDK en un proyecto de Android Studio.
2. Cómo integrar el UX en un proyecto.
3. Cómo posicionar al dron sobre un mapa de Google Maps e iniciar una misión con Waypoints.

Respecto a los tutoriales hay que destacar que, pese a que no se comenta por el fabricante, en los mismos es fundamental utilizar una versión específica de Gradle, ya que, si utilizamos las últimas versiones de éste, la aplicación nos va a dar problemas al intentar registrarla en los servidores de DJI. Desde mi punto de vista es algo que debería especificarse, claramente, porque cuando te estás iniciando en este mundo, te sientes un poco perdido y resulta frustrante no saber que la aplicación no funciona por algo tan sencillo como ésto. La versión necesaria para ejecutar la aplicación es la 3.2.1, pese a que esta versión ya tiene algunos años, no he tenido problemas en utilizar las últimas versiones de APIs como la de Google Maps o el servicio Firebase de Google.

2.3.2. Mockups de la aplicación cliente

Una vez completados los tutoriales de la documentación, mediante los cuales entendí correctamente el funcionamiento del Mobile SDK, ya podía desarrollar la aplicación del proyecto. Pero antes, es necesario desarrollar una serie de "mockups", que nos ayuden a definir cómo queremos que se utilice la aplicación, así como las funciones a implementar para cada una de ellas.
Los diseños originales de la aplicación están representados por las imágenes que se muestran a continuación:

2.3.2.1. Connection Activity

<table>
<tr><th colspan="3">Connection Activity</th></tr>
<tr>
<td> 
</td>
<td><u>Funcionalidades</u>

• Registra la aplicación en los servidores de DJI

• Valida el registro realizado

• Realiza la conexión con drones DJI

• Informa a otras vistas de la aplicación del estado de la conexión con el dron</td>
<td><u>Relaciones con “Activities”</u>

• Transmitir información a las vistas sobre los cambios en el estado de conexión entre el terminal y el dron de la marca</td>
</tr>
</table>

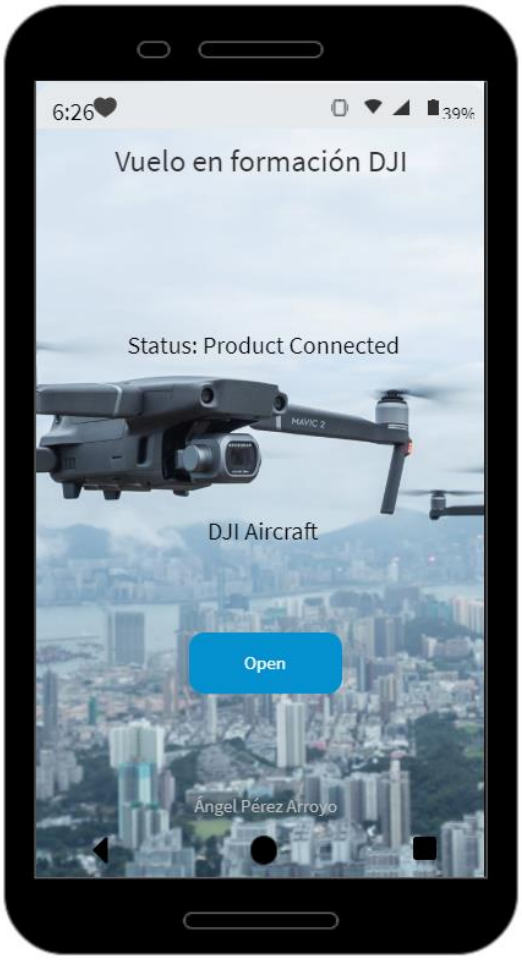


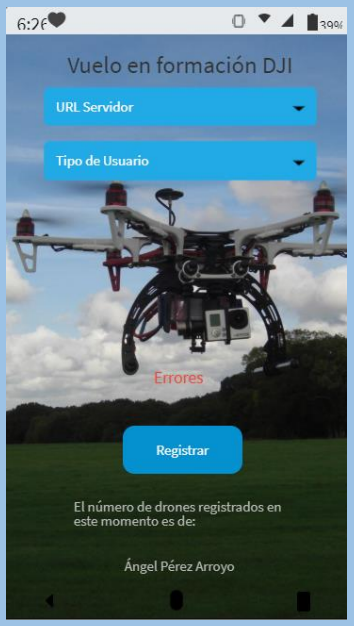
Ilustración 31. Mockup Connection Activity.

De acuerdo con lo aprendido con los tutoriales de la documentación del DJI Mobile SDK, es fundamental asegurarse que existe una conexión entre el dron y el terminal antes de realizar ninguna otra acción. En caso de no verificar esta situación, la aplicación podría llegar a fallar en 'Activities' posteriores. Por lo tanto, la primera "Activity", que diseñé servía para comprobar esto mismo.

Cuando se realiza la conexión entre ambos dispositivos el texto del "Status" cambiará de producto desconectado a producto conectado. Al mismo tiempo el mensaje donde pone "DJI Aircraft", cambiará por el nombre del dron con el que se ha establecido conexión.

Por último, para evitar que el usuario por error pase a la siguiente "Activity" sin establecer la conexión con el dron previamente, el botón que aparece en la vista diseño no se encontrará disponible. Una vez que la conexión se haya realizado y los dos mensajes superiores se hayan actualizado, el botón cambiará su estado para estar disponible y poder así, cambiar a la siguiente "Activity".

2.3.2.2. Main Activity

<table>
<tr><th colspan="3">Main Activity</th></tr>
<tr>
<td> 
</td>
<td><u>Funcionalidades</u>

• Registrar un nuevo dron en la misión

• Transmitir la petición de registro al servidor indicado por el usuario

• Informar al usuario de cualquier error ocurrido</td>
<td><u>Relaciones con “Activities”</u>

• Transmitir, en función del usuario registrado en la misión el ID del dron, ID de la misión y la URL del servidor con el que se deben realizar las siguientes peticiones API Rest.

• Si el usuario es Administrador se le trasmite la información a la “Activity” ‘Fly Registry’.

• En cambio, si el usuario no es Administrador se le trasmite la información a la “Activity” ‘Mission Activity.</td>
</tr>
</table>

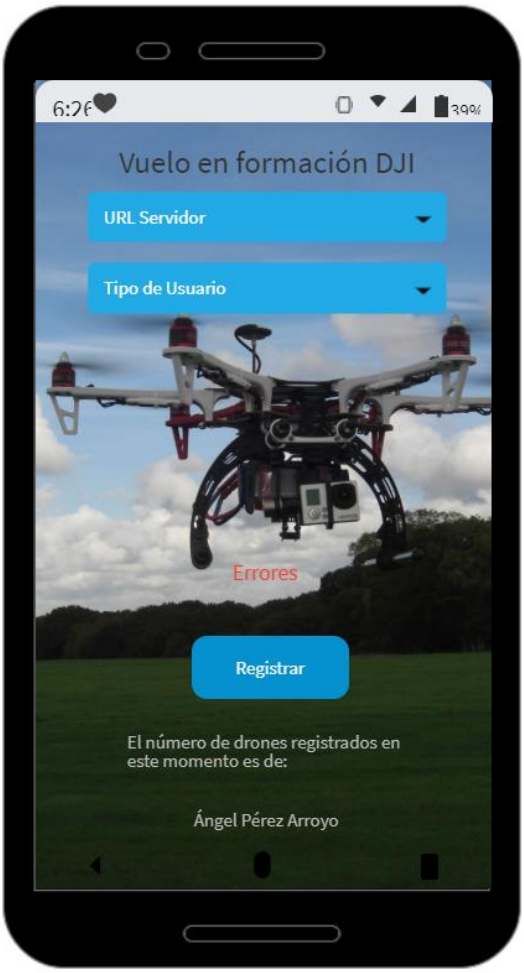


Ilustración 32. Mockup Main Activity

En esta segunda "Activity" el usuario deberá de indicar la URL del servidor a conectarse y el tipo de usuario con el que desea registrarse.

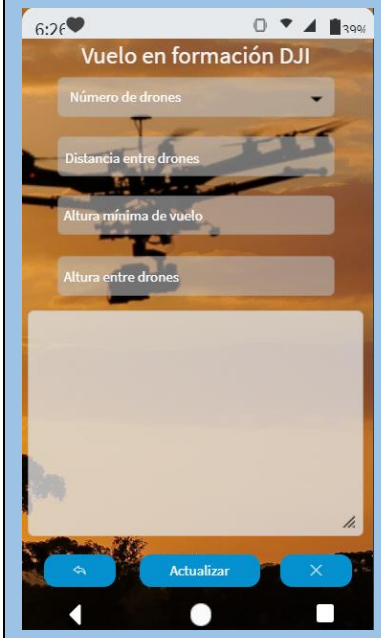
Cabe destacar que el campo URL permitirá recordar la última URL válida introducida por el usuario, de manera que, si el servidor no ha cambiado de dirección no será necesario modificar el valor de este campo.

Cuando el usuario pulse el botón "registrar", la aplicación verificará que el campo de la URL y el tipo de usuario se han completado. En caso de no haberlo efectuado, se informará al usuario a través del campo "errores" en color rojo. En cuanto a este campo, hay que destacar que se encontrará oculto y sólo será visible por el usuario, si es necesario informarle de algún error.

Además, la aplicación mostrará el número de drones registrados en el servidor en ese momento.

Al pulsar el botón se realizará una petición API Rest al servidor de la aplicación para comprobar que se puede registrar a ese dron en el sistema. En caso afirmativo, la aplicación cliente cambiará a la siguiente vista y, en caso contrario se le informará al usuario del error en el campo de errores mencionado anteriormente.

2.3.2.3. Fly Registry

<table>
<tr><th colspan="3">Fly Registry</th></tr>
<tr>
<td> 
</td>
<td><u>Funcionalidades</u>

• Permite actualizar los datos de la misión (número de drones, altura de vuelo, altura entre drones y distancia entre drones)

• Le permite al usuario de una manera sencilla dibujar la formación de drones deseada

• Permite borrar la formación dibujada las veces que sea necesario</td>
<td><u>Relaciones con “Activities”</u>

• Como el usuario debe ser de tipo Administrador para acceder a esta “Activity”, se le transmite a la ‘Mission Activity’ la URL, el tipo de usuario, el ID del dron y el ID de la misión</td>
</tr>
</table>

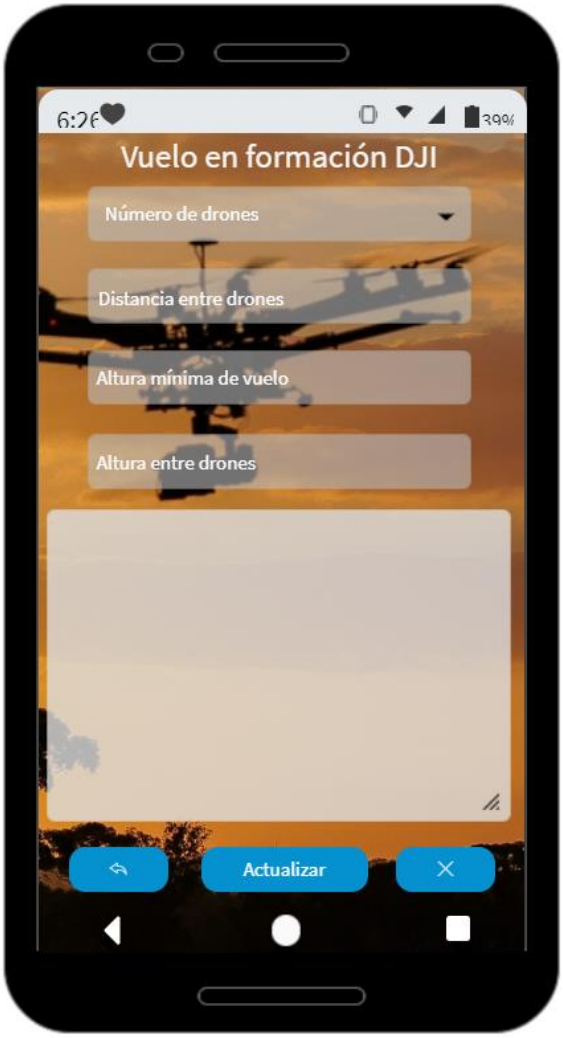


Ilustración 33. Mockup FlyRegistry Activity

En el caso de haber registrado a un usuario administrador, se le redirigirá a esta "Activity" y, en caso contrario, se le enviará a la de la misión.

En esta "Activity", el usuario administrador de la misión podrá modificar el número de drones que se pueden registrar, la distancia que tiene que existir entre ellos, la altura mínima de vuelo y la altura entre los distintos drones. Además, al final de la "Activity" el usuario Administrador podrá dibujar de una manera sencilla la formación que desea que los drones mantengan, así como la distancia al centro del Waypoint.

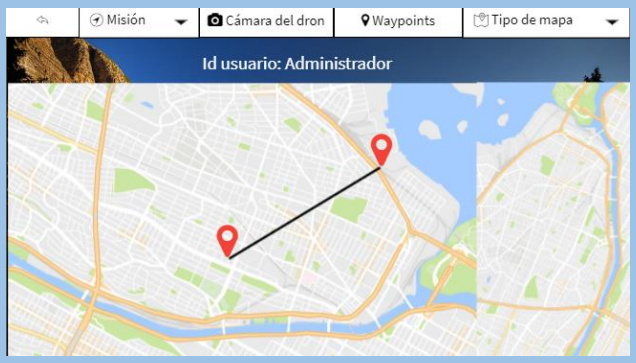
Por último, los tres botones inferiores nos permitirán: borrar el último dron dibujado por el usuario, eliminar todos los drones dibujados o actualizar los datos de la misión.

Si el usuario pulsa el botón de atrás, la aplicación eliminará el último dron dibujado. Por supuesto, se podrá pulsar tantas veces el usuario considere oportuno para borrar tantos drones como considere.

En cambio, si se pulsa el botón de la 'X' borrará todos los drones dibujados.

Finalmente, si el usuario pulsa el botón "actualizar", el cliente generará una petición API Rest al servidor para enviarle la nueva configuración que debe tener la misión, ya que cuando se crea ésta siempre tiene unos valores por defecto, que durante esta "Activity" se modifican. Además, al darle al botón se comprueba, previamente, que todos los campos han sido completados por el usuario.

2.3.2.4. Mission Activity

<table>
<tr><th colspan="2">Mission Activity</th></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td><u>Funcionalidades</u>

• Ejecutar, pausar o detener misiones

• Añadir Waypoints y enviarlos al servidor

• Acceder a la cámara del dron

• Acceder a la vista específica para verificar si se pueden volar drones donde nos encontramos

• Cambiar el tipo de mapa

• Descargar los Waypoints específicos para el dron conectado al terminal</td><td><u>Relaciones con “Activities”</u>

• Se trata de la última y principal “Activity” de la aplicación. En esta vista será donde se reciban todos los datos introducidos por el usuario para poder ejecutar las misiones</td></tr>
</table>

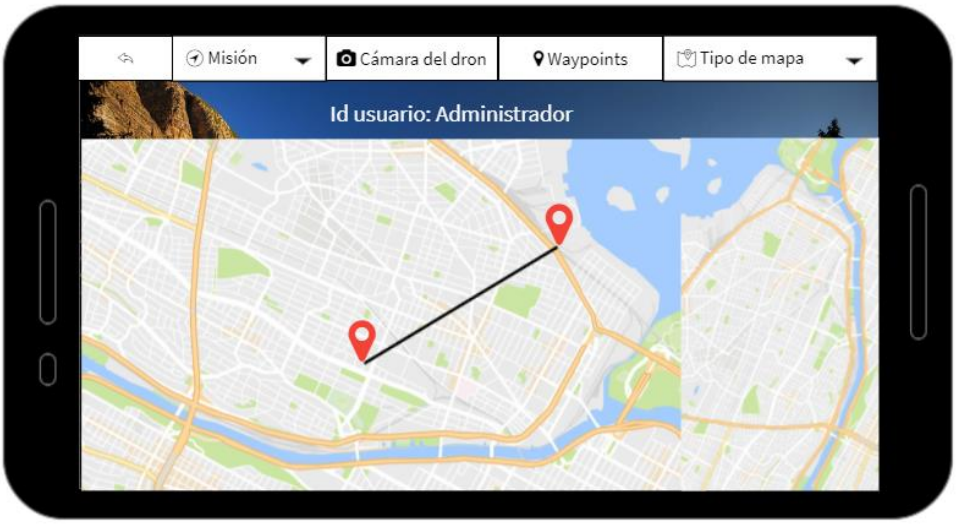


Ilustración 34. Mockup Mission Activity

Para esta vista se generarán dos variantes, una para el Administrador la cual contendrá todas las funcionalidades y otra para los usuarios, que tendrá una vista simplificada.

La finalidad de la aplicación es que desde un móvil se pueda ejecutar la misión de todos los drones sin problema, pero debido a las características del Mobile SDK funciones como la de acceder a la cámara del dron tienen que hacerse desde el propio dispositivo conectado al dron. Por lo tanto, esta funcionalidad se mantendrá incluso en los usuarios no administradores.

Desde esta vista, pulsando la opción misión, el Administrador podrá ejecutarla y controlar su desarrollo. Esto es debido a que en el momento en el que el usuario presione el botón de "iniciar misión" todos los dispositivos recibirán una notificación indicando que deben iniciar la misión. De esta manera, todos los drones iniciarán la misión al mismo tiempo.

El botón de los Waypoints servirá para que cada terminal pueda solicitar al servidor los Waypoints referentes a su dron.

Por último, también existirá la posibilidad de cambiar el tipo de mapa que se muestra en la pantalla.

2.3.2.5. UX Activity

UX Activity	
	
<u>Funcionalidades</u> • Permite visualizar lo que está viendo el dron con su cámara • Acceso al porcentaje de carga de la batería, número de satélites conectados o calidad de la señal recibida por el dron • Distancia actual entre la posición del dron y la del control remoto • Posibilidad de ejecutar vídeos o fotos de manera manual • Permite visualizar la velocidad y altura de vuelo en cada momento	<u>Relaciones con "Activities"</u> • Vista a la que se puede acceder desde la "Mission Actitviy" por cualquier usuario de la aplicación.

Ilustración 35. Mockup UX Activity

Tal y como se ha indicado previamente, el Mobile SDK incluye un paquete llamado UX, que será el utilizado para construir una vista como la de la aplicación DJI Go 4.

Esta "Activity" será prácticamente igual que la de referencia de la marca, pero con la salvedad de que no se encontrarán disponibles las opciones de despegue y aterrizaje automáticos. Esto es debido a que se quiere evitar que por error algún usuario los pulse y afecte al resultado de la misión.

Al mismo tiempo lo que se permite con esta "Activity" es que cualquiera de los usuarios pueda grabar o tomar fotos con la cámara del dron de una manera muy intuitiva, durante la ejecución de la misión.

2.3.2.6. Comunicación entre terminales Android

Debido a la problemática de tener que conectar un terminal Android a cada uno de los drones DJI, resulta imprescindible establecer un proceso de comunicación, que de manera sencilla permita enviar desde un solo cliente, notificaciones al resto de los terminales de la misión. Por ello, cada vez que el usuario Administrador desee iniciar, pausar/reanudar o detener la misión, la aplicación generará una notificación para informar al resto de los clientes de la acción, que deben realizar sobre el dron conectado a ellos.

Este proceso se puede ver de una manera más desarrollada en el siguiente diagrama.

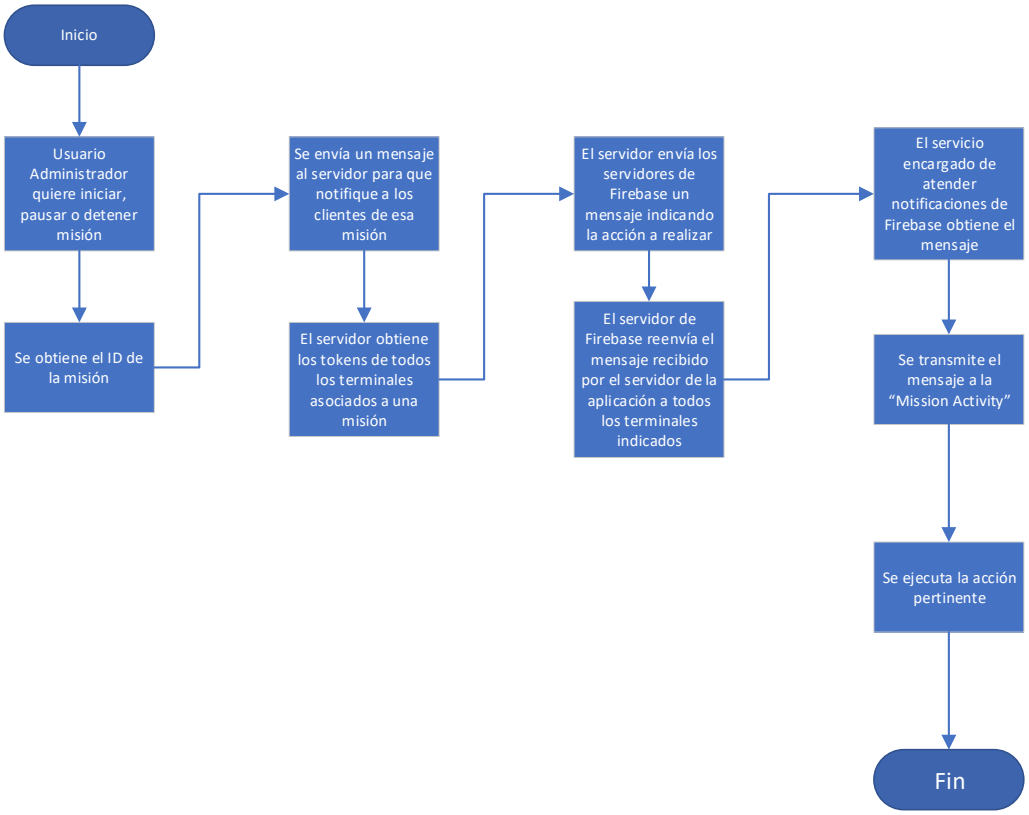


Ilustración 36. Proceso de comunicación entre clientes

2.4. Diseños finales

En el presente apartado pasaremos a ver los diseños finales, así como las pequeñas variaciones que se han producido con respecto a los "mockups" planteados en el apartado anterior y se procederá a explicar el motivo que ha llevado a realizar dichos cambios.

La primera vista que aparece al abrir la aplicación es la siguiente:

Ilustración 37. Connection Botón Activado

Ilustración 38. Connection Botón Desactivado

En ella tal y como se ha indicado en el apartado previo, esta vista inhabilitará el funcionamiento del botón hasta que la aplicación no se haya registrado correctamente y no se haya establecido conexión con algún dron compatible.

Además, ambos mensajes cambiarán para indicar que existe una conexión con un dron DJI y el modelo de éste. Una vez se hayan modificado ambos textos, el color del botón cambiará automáticamente y se encontrará disponible para que el usuario pueda presionarlo y pasar a la siguiente "Activity".

A diferencia de la "Activity" anterior, el diseño de ésta no estaba contemplado, en principio, pero al ver la necesidad de poder crear alguna misión por si existieran múltiples grupos de usuarios, que quisieran realizar misiones diferentes en lugares distintos, se decidió realizar la siguiente "Activity" con la finalidad de solucionar dicha circunstancia no tenida en cuenta inicialmente.

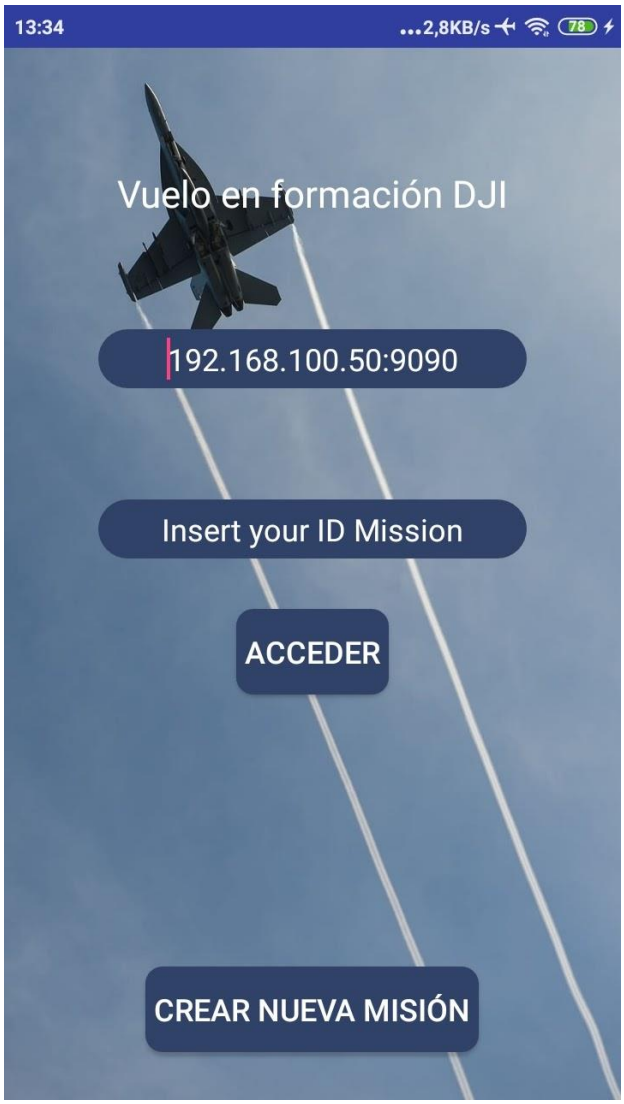


Ilustración 39. Create Mission Activity

Al disponer de la opción de crear una nueva misión se decidió mover el campo de introducir la URL donde se encuentra el servidor a esta nueva ventana.

Además, tal y como se puede apreciar en la imagen de la izquierda, se han creado dos secciones dentro de la "Activity". La primera de ellas permite a un usuario acceder a una misión. Este paso sería el necesario para que, una vez se haya creado una misión desde un cliente, el resto puedan acceder fácilmente.

Cabe destacar que, al pulsar el botón de acceder a la misión X se validará que el campo se encuentre completado por parte del usuario.

En el caso de no disponer de una misión y necesitar crear una, se ha incluido un botón en la parte inferior de la pantalla con ese mismo propósito.

Además, deberemos de tener en cuenta que se han contemplado los posibles problemas que, pudieran surgir tanto al registrar una nueva misión o al comprobar si la introducida por el usuario existe. Para ello, se le notificará al usuario del error en particular ocurrido durante este tipo de procesos vía un 'Toast'.

Por último, hay que destacar que al crear una nueva misión le aparecerá al usuario un 'Toast' en el que se le indique el ID de la nueva misión. De esta manera, podrá conocer el ID que se tiene que introducir en el resto de los terminales Android, para poder añadirlos a la misión.
Al haberme visto obligado a mover el campo de la URL del servidor a la anterior "Activity", por ser estrictamente necesario, ésta se ha quedado con la responsabilidad de registrar a los distintos drones que van a intervenir en la misión.

Ilustración 40. Main Activity

Tal y como se puede apreciar en la imagen de la izquierda, el usuario dispone de un desplegable que le permite seleccionar entre "Administrador" y "Usuario".

Al pulsar el botón "Registrar" se comprobará si se ha introducido el tipo de usuario. De no ser así, lo que hará es informar que debe introducir uno.

Si se ha introducido el tipo de usuario se realizará una petición API Rest al servidor de la aplicación con el tipo de usuario seleccionado.

En el caso que, no exista ningún problema al registrar el dron conectado a ese terminal móvil, se le enviará a la siguiente "Activity", en función de si éste es Administrador o Usuario normal.

Si llegara a ocurrir algún tipo de fallo, se le informaría a través del campo de error que se ha podido ver en el Mockup de esta "Activity".

Dicho campo de error no sale en la imagen superior porque, como se ha indicado previamente, ese campo tan sólo se vuelve visible cuando existe algún error. Además, el contenido del mensaje varía en función del tipo de error ocurrido.

Si no ha surgido ningún error durante el registro del nuevo dron, llegaríamos a la siguiente ventana en el caso de ser Administradores.

Ilustración 41. Fly Registry sin Datos

Ilustración 42. Fly Registry con Datos

A través de esta "Activity" podremos actualizar los parámetros de la misión todas las veces que consideremos necesario. Cabe destacar que esta "Activity" es de suma importancia ya que cuando se crea una misión se le dan unos parámetros por defecto, y es aquí, donde el usuario Administrador puede cambiar dichos valores de una manera muy sencilla e intuitiva.

Además, como se puede ver en las imágenes superiores, cada vez que el usuario marca una opción del desplegable, aparece un pequeño 'Toast' informándole de la opción que acaba de seleccionar.

En la figura número 42 aparece como se pueden dibujar las formaciones que llevarán los drones de una manera sencilla. Para hacerlo, tan sólo bastará con seleccionar el número de drones que deseamos que puedan registrarse para la misión. Una vez realizado ese proceso bastará con tocar con el dedo sobre el punto en el que deseemos que se coloque el dron en la formación. Además, se dibujará una línea blanca entre el icono y el centro del operativo, que simboliza el Waypoint o coordenada GPS introducida por el usuario.

Por último y pese a que no aparezca en las imágenes superiores, los tres botones inferiores que sí aparecían en el mockup se han añadido en la versión final de la “Activity”.

El motivo por el que no aparecen en la foto es por el tamaño de pantalla del móvil empleado para realizar las capturas. Para solucionar la problemática que pudieran llegar a tener los propietarios de terminales Android con un tamaño de pantalla y resolución insuficiente para mostrar todo el contenido de esta “Activity”, se decidió poner a todos los elementos dentro de un ‘Scroll View’, el cual nos permite subir y bajar la pantalla para el caso de no poder mostrar todo el contenido en la misma. De esta manera, usuarios con distintas resoluciones y tamaños de pantalla podrían llegar a utilizar la aplicación sin problemas. Además, cabe destacar que al verificar el problema que surgía en esta “Activity” se modificaron todas, para que ninguna pudiera resultar compleja o imposible de utilizar por parte de ciertos usuarios.

Entonces una vez hemos deslizado la pantalla hacia abajo, ya sí podemos comprobar que los botones se han mantenido tal y como se indica en la siguiente imagen:

Ilustración 43. Fly Registry Inferior

Por último, si el proceso a través de todas y cada una de las distintas "Activities" anteriores ha sido el correcto, deberíamos llegar a la "Activity" principal de la aplicación. Mediante ésta podremos controlar el correcto desarrollo de la misión, así como el progreso de ésta, entre otras posibles utilidades.

Tal y como se ha indicado anteriormente, para las funcionalidades de esta "Activity" se van a ofrecer distintas vistas según el tipo de usuario registrado en la misión. Unas serán para los usuarios comunes, que accederán a una vista simplificada y, por lo tanto, no podrán ejecutar la misión desde su terminal Android, ya que todo lo relacionado con ésta se encuentra limitado a los usuarios Administradores.

Otro de los factores en el que ve limitada su funcionalidad es a la hora de añadir los Waypoints, esta opción está reservada a los usuarios Administradores.

Las funciones principales que sí están a disposición de todos los usuarios son las siguientes: acceder a la información del dron conectado al dispositivo, descargar los Waypoints a ejecutar por el dron conectado y cambiar el tipo de mapa.

Ilustración 44. Mission Activity Usuario

Al pulsar el botón de "Información Dron" lo que haremos será acceder a la vista que tiene prácticamente las mismas funciones que la aplicación DJI Go 4, tal y como se ha indicado en el punto 2.2 y cuyo resultado final es el siguiente:

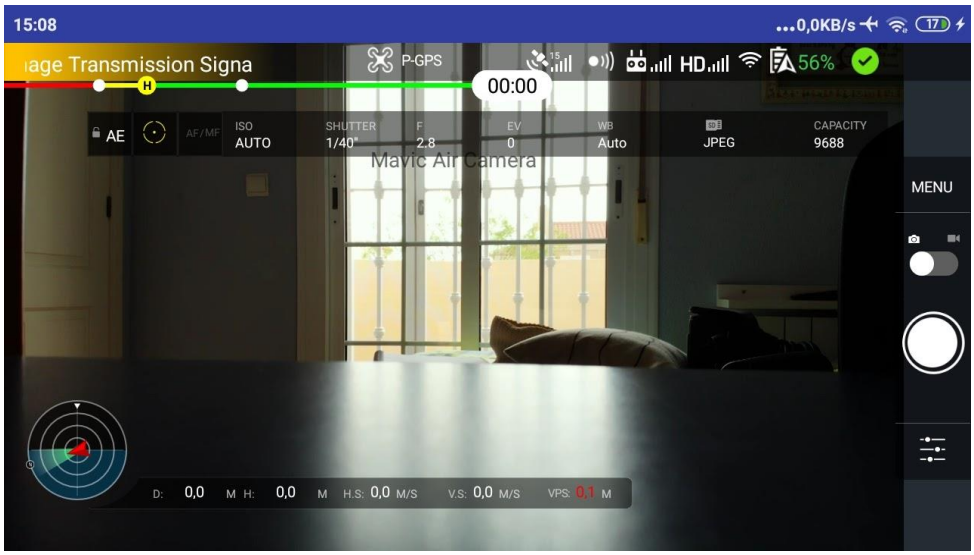


Ilustración 45. UX Activity

Algunas de las funciones que pueden resultar prácticas en esta "Activity" son, por ejemplo, visualizar el estado de la batería, la calidad de la señal, si algún sensor del dron necesita una calibración, etc. Además, tal y como se ha indicado en el apartado anterior también podremos realizar fotos o vídeos y visualizarlos.

Es por ello, que esta "Activity" puede resultar de gran utilidad, al permitirnos comprobar el estado de cada uno de los drones, para asegurarnos que no va a ocurrir ninguna contingencia durante el desarrollo de la misión, que nos obligue a detenerla.

El botón de Waypoints que aparece en la vista de usuario sirve para descargar los puntos a los que ha de acceder el dron conectado a ese dispositivo móvil. Al pulsar dicho botón la aplicación preparará una petición API Rest al servidor solicitando los puntos GPS donde están los Waypoints propios del dron. En cuanto al proveedor del mapa utilizado hay que destacar que es de Google Maps, al haber empleado el Google Maps SDK para Android con la finalidad de poder integrarlo.

Una vez el cliente recibe dichos puntos GPS los pinta en el mapa de la siguiente manera:

Ilustración 46. Mission Activity Waypoints Usuario

Tal y como se puede apreciar en la imagen superior si pulsamos sobre cualquiera de los marcadores dibujados sobre el mapa, éste nos indica el Waypoint de forma exacta. De esta manera podremos comprobar de un simple vistazo si los puntos se han obtenido correctamente. Por último, destacar que, puede verse en la imagen superior los distintos Waypoints unidos con líneas rojas, que nos ayudarán a determinar el camino que tendrá que seguir el dron que está conectado al móvil para ir de un punto GPS a otro.

Finalmente, me gustaría destacar que tal y como se puede apreciar en la imagen, la aplicación te geolocaliza para que no tengas que buscar tu ubicación, sino que una vez obtiene tu coordenada GPS actual carga el mapa de Google Maps, mueve la cámara y hace zoom hasta tu posición. De esta manera el usuario no tiene que molestarse en ubicarse sobre el mapa, lo cual puede resultar muy práctico si se van a realizar misiones en distintos lugares del mundo.

Además, si por cualquier motivo el usuario se desplaza mucho por el mapa y no supiera luego encontrar su posición, pulsando sobre el botón superior derecho situado dentro del mapa, la aplicación movería la cámara del mapa hasta su ubicación de manera inmediata.

En cuanto al botón de “Tipo de mapa” nos permite abrir la siguiente ventana en la cual de una manera rápida podremos cambiar la vista de mapa entre las tres opciones distintas que se proporcionan por parte de la aplicación. En referencia al tipo de mapa que se carga por defecto destacar que es el de vista satélite, y desde mi punto de vista es el que podría resultar más claro, aunque el usuario podrá cambiar de una vista a otra de mapa de una manera sencilla.

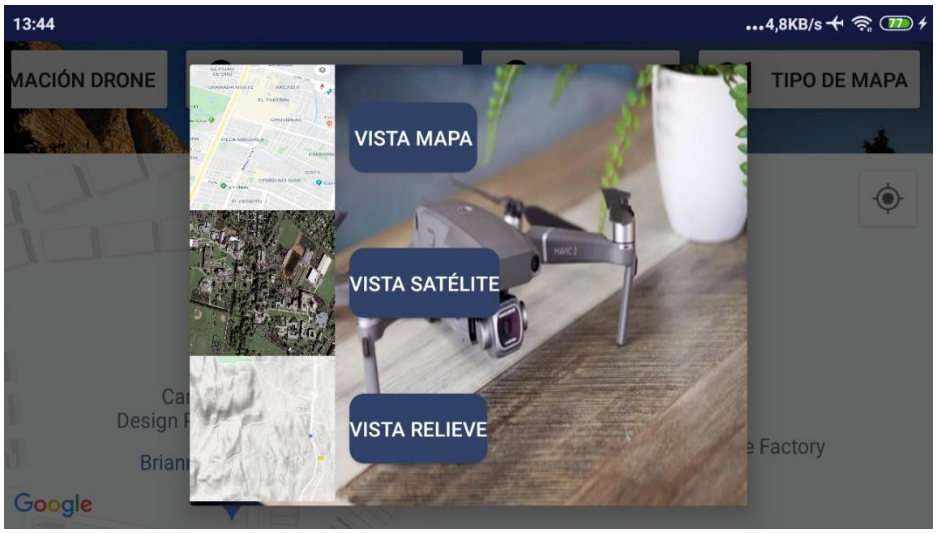


Ilustración 47. Cambio de mapa

Finalmente, hay que añadir que en el momento en el que el dron de la marca DJI utilizado obtenga sus coordenadas GPS, la aplicación de manera automática dibujará un avión de color rojo sobre el mapa indicando la posición actual del dron. Además, será este mismo icono el que nos permitirá determinar la posición en cada momento de cada uno de los drones, ya que estos iconos se irán moviendo sobre el mapa de Google Maps indicando la posición.

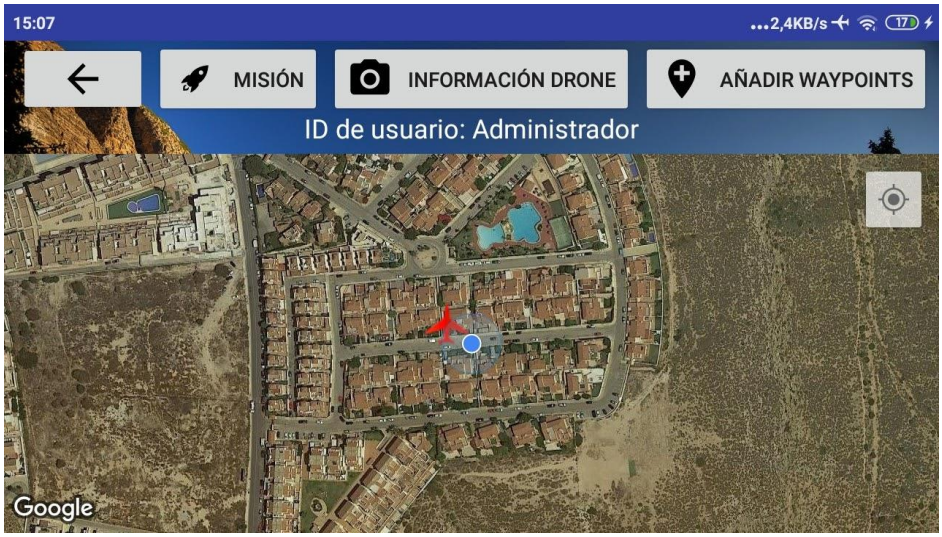


Ilustración 48. Dron sobre el mapa Usuario

Ahora podemos ver el diseño de la vista para los usuarios Administradores, que es la siguiente:

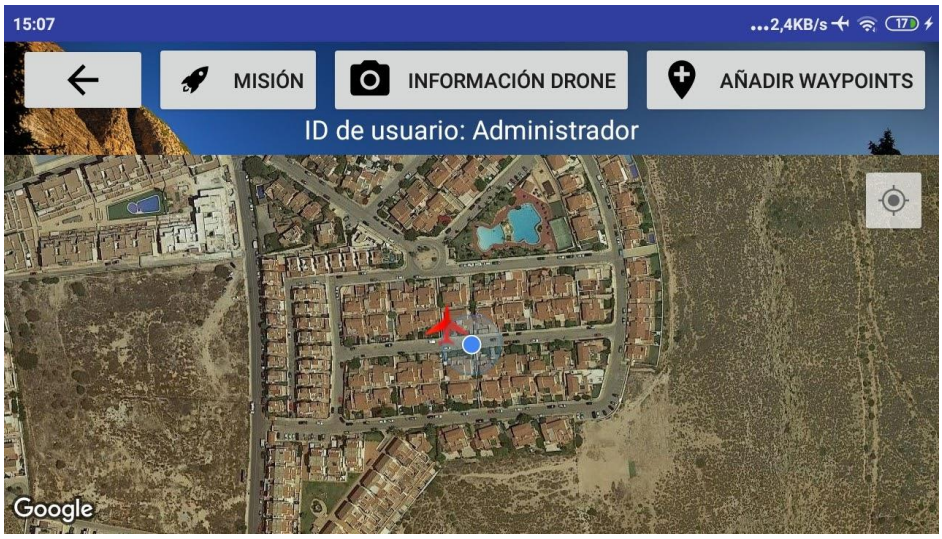


Ilustración 49. Dron sobre el mapa Administrador

Tal y como podemos ver, la posición del dron sobre el mapa se nos muestra de igual manera para todos usuarios.

El primer botón diferente respecto a la vista de usuarios no administradores es el botón con una flecha hacia la izquierda. Si el usuario pulsara este botón volvería automáticamente a la vista anterior, es decir, en su caso volvería a la vista para actualizar los parámetros de la misión. Este botón puede resultar muy práctico si se desean añadir nuevos drones o modificar la altura de vuelo de éstos.

En referencia al botón de la misión, hay que destacar que es la opción principal de esta "Activity", y con éste el usuario Administrador de la misión podrá iniciar, pausar o reanudar y detener la misión de todos los drones en cualquier momento.

Al pulsar este botón "Misión" al Administrador le aparecerá la siguiente ventana:

Ilustración 50. Layout Misión

Nada más cargar la vista de esta nueva ventana, podremos apreciar que en la parte inferior dispondremos de los tres botones necesarios para controlar la misión y, por lo tanto, el comportamiento de todos y cada uno de los drones empleados para la ejecución de la misión.

En cuanto al funcionamiento de estos botones es muy simple, su única utilidad es la de enviar al servidor una solicitud para que éste, remita a todos los miembros de la misión una notificación, a través de Firebase a cada uno de los dispositivos Android. Dicha notificación contendrá la información necesaria para que la aplicación cliente sepa el mensaje a enviar a cada uno de los drones.

En la parte superior izquierda encontramos el botón de "GEO SYSTEM". Mediante este botón podremos acceder a una nueva "Activity" que nos permitirá visualizar de una manera gráfica si nos encontramos en una zona en la que no se puede volar o si los drones durante la misión entrarían en una zona sobre la que no es legal el vuelo de este tipo de aeronaves.

Una vez cargamos esa nueva "Activity" veremos lo siguiente:

Ilustración 51. GEO Activity

Nuevamente la aplicación nos volverá a geolocalizar para mover la cámara del mapa hasta nuestra posición. En la parte izquierda aparecerán los tres botones necesarios para obtener la información de zonas de vuelo válidas.

Para ello, el primer paso es iniciar sesión con nuestro usuario y contraseña de DJI. Esto es necesario, debido a que la base de datos de vuelo seguro que vamos a utilizar es la propia de la marca y nos exige autenticación antes de poder acceder a ella.

Por lo tanto, iniciamos sesión tal y como se puede ver en la imagen inferior:

Ilustración 52. GEO Activity Login

Una vez iniciemos la sesión correctamente volveremos nuevamente al mapa. Una vez allí, ya nos encontraremos en disposición de poder utilizar el botón "GET SURROUNDING NFZ". Dicho botón comprobará que nos hayamos autenticado correctamente antes de dibujar sobre el mapa las zonas de vuelo restringidas a este tipo de drones tal y como puede verse en la siguiente imagen.

Ilustración 53. GEO Activity Zones Printed

Como se puede ver en la imagen superior, debido a la cercanía de mi ubicación con el aeropuerto de Alicante-Elche la aplicación me estaría mostrando las zonas en las que no sería legal volar los drones o despegarlos.

Para salir de esta "Activity" y volver a la de la misión debemos pulsar el botón con el símbolo de atrás. En cuanto al botón de "Logout" sirve para cerrar sesión en el caso de que así lo deseemos.

Una vez hayamos salido de la anterior "Activity" volveremos al punto donde nos encontrábamos antes de cambiarla:

Ilustración 54. Mission Layout 2

En cuanto a la opción central que es la de "ENVIAR WAYPOINTS" será la que nos permita transmitir los Waypoints introducidos al servidor. Por ello, al pulsar este botón la aplicación comprobará que tengamos al menos un Waypoint introducido y que la velocidad de la misión se ha establecido. Para introducir dicha velocidad debemos utilizar el desplegable que está en la parte superior derecha de esta ventana.

Si se llegan a cumplir todas y cada una de las condiciones, el cliente generará una nueva petición API Rest al servidor de la aplicación donde se enviarán todos los puntos GPS introducidos por el usuario. Si el resultado de la petición es correcto y, por lo tanto, no se ha generado ningún error se le informará al usuario de ello y se borrarán los marcadores dibujados sobre el mapa de Google Maps de la aplicación.

En caso de que surgiera algún inconveniente o problema se le informaría al usuario de ese error en particular.

Una vez el servidor ha recibido los puntos y los ha guardado en la base de datos, se pondrá a calcular todos y cada uno de los puntos GPS relativos a cada Waypoint introducido por el usuario administrador. Todo esto se realiza con la finalidad de que todos los drones puedan disponer de puntos GPS distintos que les permita mantener la formación en todo momento.

En cuanto a cómo añadir Waypoints sobre el mapa para poder enviarlos al servidor es muy sencillo, debido a que existe una opción específicamente destinada para ello.

Si nos fijamos en la parte superior de la "Activity" existe un botón "AÑADIR WAYPOINTS", al presionar dicha opción la aplicación se encontrará en disposición de añadir nuevos Waypoints sobre el mapa. En caso de no activar esta opción es imposible añadir Waypoints y se ha diseñado así para evitar introducirlos a la misión por error.

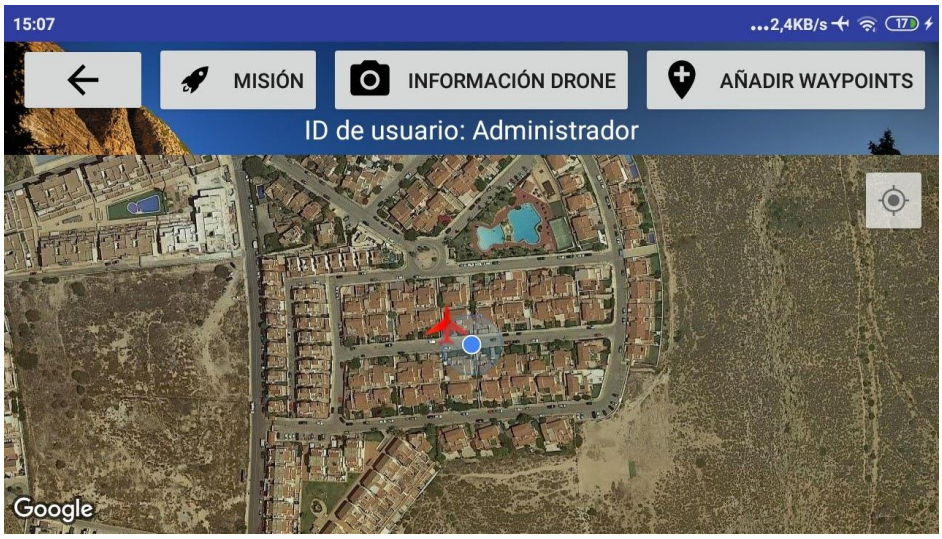


Ilustración 55. Mission Activity Add Waypoints

Por lo tanto, y tal y como se puede ver en la imagen siguiente, podremos añadir tantos Waypoints como queramos, teniendo en cuenta que siempre deberemos añadir mínimo un Waypoint para la misión y un máximo de 99, ya que este último es el número máximo de coordenadas que se le pueden pasar a un dron DJI por misión.

Ilustración 56. Mission Activity Adding Waypoints

Debemos tener en cuenta que al presionar el botón para añadir Waypoints éste desaparecerá, y aparecerán dos nuevos que servirán para salir del modo añadir Waypoints y para poder borrar todos los Waypoints del mapa.

2.5. Arquitectura de la aplicación

Debido a la problemática de tener que establecer algún mecanismo de comunicación entre distintos terminales Android, para poder controlar una misión desde un único dispositivo, se decidió adoptar una arquitectura cliente-servidor.

Esta es un modelo de diseño software en el que un cliente, siendo, en este caso, la aplicación en Android realiza una serie de peticiones a otro programa alojado en un equipo distinto (servidor). El programa ejecutado en el servidor se encargará de proporcionar al cliente respuestas a sus peticiones, siendo las HTTP las empleadas para el desarrollo de este proyecto.

Por lo tanto, tal y como se puede apreciar en la ilustración 57, los terminales se encuentran conectados al radio control para mandar y recibir información del dron y, al mismo tiempo a Internet.

La conexión con el control remoto resulta fundamental, al ser necesaria para enviar al dron los distintos Waypoints a los que tendrá que acudir, entre otros datos necesario para el correcto desarrollo de la misión

Por último, debido a la necesidad de conectarse a servidores para poder enviar y recibir mensajes resultará imprescindible la existencia de un tráfico de datos.

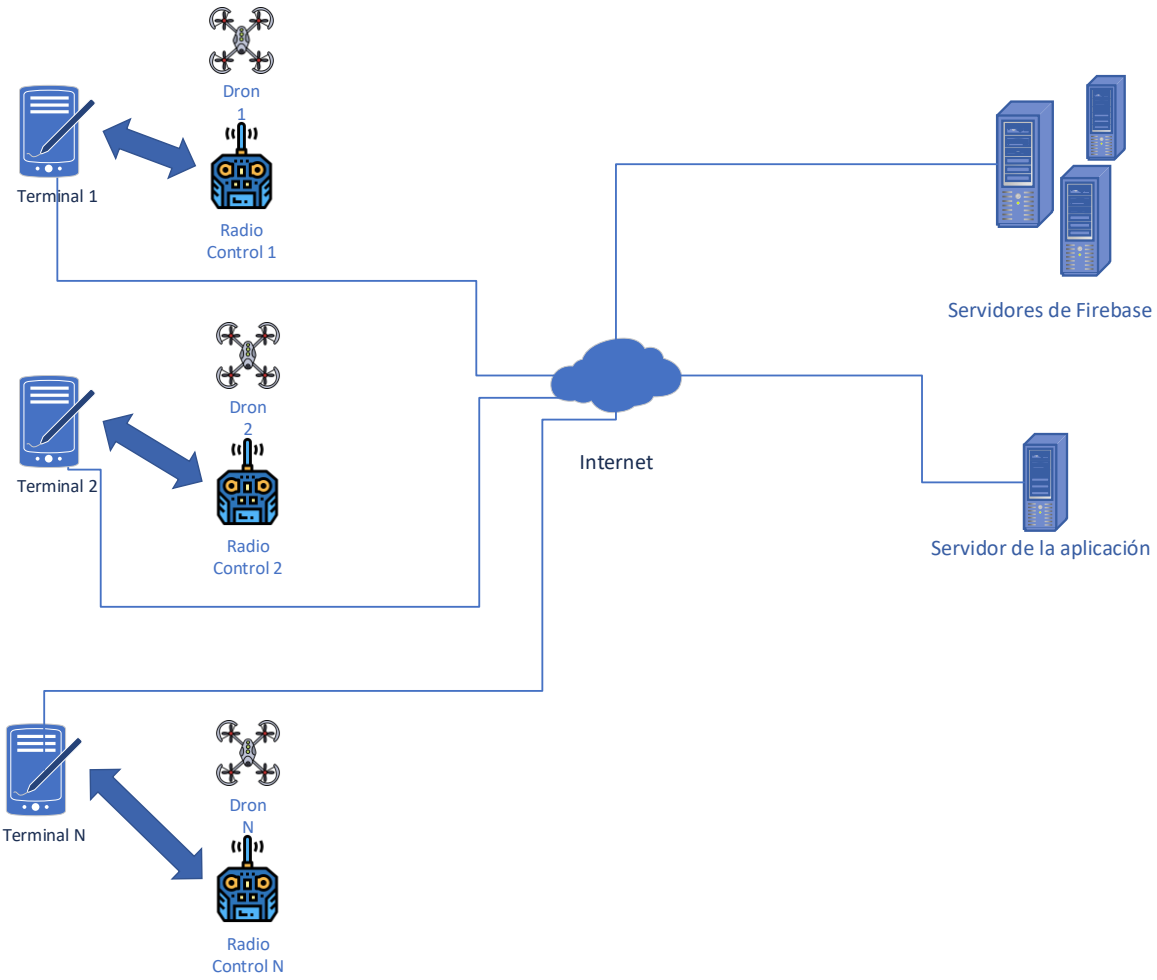


Ilustración 57. Diagrama Cliente-Servidor

2.6. Desarrollo de la aplicación

En este último apartado centrado en el desarrollo del Trabajo Final de Grado, analizaremos el funcionamiento tanto del cliente como del servidor, mediante el uso de unos diagramas de flujo, que nos permitirán ver de una manera mucho más sencilla cómo está desarrollada internamente la aplicación y todos sus procesos. Además, también se mostrarán imágenes de cómo se han implementado los principales procesos de cada una de las "Activities", así como las funciones encargadas de atender las peticiones HTTP generadas por la aplicación móvil.

2.6.1. Aclaración de las funcionalidades principales del cliente

En referencia a las cuestiones que se van a analizar en apartados posteriores dentro del punto 2.5 tenemos las siguientes:

- En el apartado 2.5.1.1 (Connection Activity) se procederá a analizar el proceso de registro de la aplicación en los servidores de DJI, y cómo se lleva a cabo el proceso para establecer conexión entre el dron y el terminal Android utilizado.
  Ambos procesos resultan cruciales para el correcto funcionamiento de la aplicación, debido a que no podremos utilizar elementos de la API del DJI Mobile SDK sin que ésta se encuentre correctamente registrada. En el caso contrario, ésta simplemente se cerrará automáticamente. En cuanto al proceso de conexión entre el dron y el terminal, es fundamental la correcta conexión entre ambos dispositivos para poder controlar el vehículo desde el terminal.

- En el apartado 2.5.1.2 (Create Mission) se analizará el proceso de creación de una nueva misión o acceso a una ya creada, tanto en el cliente como en el servidor.

- En referencia al apartado 2.5.1.3 (Main Activity) se expondrá el proceso realizado tanto en la aplicación cliente como en el servidor, para registrar a un usuario. Este paso es clave, en cuanto determina el usuario Administrador de la misión.

- En cuanto al apartado 2.5.1.4 (Fly Registry) será una vista tan sólo accesible para los usuarios administradores de la misión, y dará acceso a las funcionalidades necesarias para personalizar la misión pudiendo definir: altura mínima de vuelo, velocidad empleada, Waypoints a realizar o la formación de los drones durante la misión.

- Por último, en el apartado 2.5.1.5 (Mission Activity) tal y como se ha venido mencionando en apartados previos, será la vista principal de la aplicación cliente, que comprobará el tipo de usuario para ofrecerle unas determinadas funcionalidades. Desde esta vista el usuario administrador tendrá acceso a un amplio abanico de posibilidades como: ejecutar misiones, añadir Waypoints, modificar el tipo de mapa o descargar los Waypoints pertenecientes al dron conectado a ese dispositivo móvil.

2.6.1.1. Connection Activity

Al abrir la aplicación ésta intentará conectarse a los servidores de DJI para comprobar y registrar que la aplicación está validada. Por lo tanto, es fundamental que el dispositivo Android sobre el que se vaya a ejecutar la aplicación disponga de conexión a Internet ya sea vía Wi-Fi o vía Datos Móviles.

En función del resultado obtenido de ese registro la aplicación informará al usuario de cualquier error ocurrido o se intentará conectar al dron para habilitar el botón inferior, tal y como se ha expuesto en el apartado previo.

En cuanto al método principal de esta "Activity" encontramos que es el llamado "startSDKRegistration". Dicho método comprueba si el proceso de registro de la aplicación Android se ha iniciado. En caso contrario se genera una tarea asíncrona, con el objetivo de no bloquear la interfaz hasta que ésta está completada, el cual, dependiendo de la conexión a Internet disponible en ese momento y del terminal sobre el que se está ejecutando la aplicación, puede llegar a tardar varios segundos en completarse.

Una vez iniciada la tarea asíncrona, el primer paso que se lleva a cabo es el de registro de la aplicación, si se ha realizado correctamente se conectar el terminal con el dron. En caso contrario, se le indica al usuario que la aplicación no ha podido llevar a cabo la tarea de registro.

Por último, hay que destacar que, este método también nos permite controlar mediante el uso de "onProductDisconnect" y "onProductConnect" si el terminal Android se desconecta del mando en cualquier momento.

```
 * Method to register the application.
 */
private void startSDKRegistration()
{
    //Si el proceso de registro no se ha llevado acabo entra
    if (isRegistrationInProgress.compareAndSet( expect: false,  update: true))
    {
        AsyncTask.execute(new Runnable()
        {
            @Override
            public void run() {
                showToast(  toastMsg: "registering, pls wait...");
                DJISDKManager.getInstance().registerApp(getApplicationContext(),
                        new DJISDKManager.SDKManagerCallback()
                {
                    /*
                        If the registration is successful, invoke the startConnectionToProduct()
                        method of DJISDKManager inside the onRegister() callback method to start
                        the connection between SDK and the DJI Products.
                     */
                    @Override
                    public void onRegister(DJIError djiError)
                    {
                        if (djiError == DJISDKError.REGISTRATION_SUCCESS)
                        {
                            DJILog.e( tag: "App registration",
                                    DJISDKError.REGISTRATION_SUCCESS.getDescription());
                            DJISDKManager.getInstance().startConnectionToProduct();
                            showToast( toastMsg: "Register Success");
                        }
                        else
                            showToast(  toastMsg: "Register sdk fails, check network is available"

                        Log.v(TAG, djiError.getDescription());
                    }

                    @Override
                    public void onProductDisconnect()
                    {
                        Log.d(TAG,  msg: "onProductDisconnect");
                        showToast( toastMsg: "Product Disconnected");
                    }

                    @Override
                    public void onProductConnect(BaseProduct baseProduct)
                    {
                        Log.d(TAG, String.format("onProductConnect newProduct:%s", baseProduct))
                        showToast( toastMsg: "Product Connected");
                    }
```

Ilustración 58. Connection Activity 1

```
                @Override
                public void onComponentChange(BaseProduct.ComponentKey componentKey,
                                              BaseComponent oldComponent,
                                              BaseComponent newComponent)
                {

                    if (newComponent != null)
                    {
                        newComponent.setComponentListener(new BaseComponent.ComponentListener()
                        {
                            @Override
                            public void onConnectivityChange(boolean isConnected)
                            {
                                Log.d(TAG,  msg: "onComponentConnectivityChanged: " +
                                        isConnected);
                            }
                        });
                    }
                    Log.d(TAG,
                            String.format("onComponentChange key:%s, oldComponent:%s, " +
                                          "newComponent:%s",
                                    componentKey,
                                    oldComponent,
                                    newComponent));
                }

                @Override
                public void onInitProcess(DJISDKInitEvent djisdkInitEvent, int i)
                {

                }

                @Override
                public void onDatabaseDownloadProgress(long l, long l1)
                {

                }
            });
        }
    });
}
}
```

Ilustración 59. Connection Activity 2

Por último, se procede a mostrar a modo de resumen el funcionamiento de esta “Activity” y de la función que atiende la petición en el servidor mediante en el siguiente diagrama de flujo.

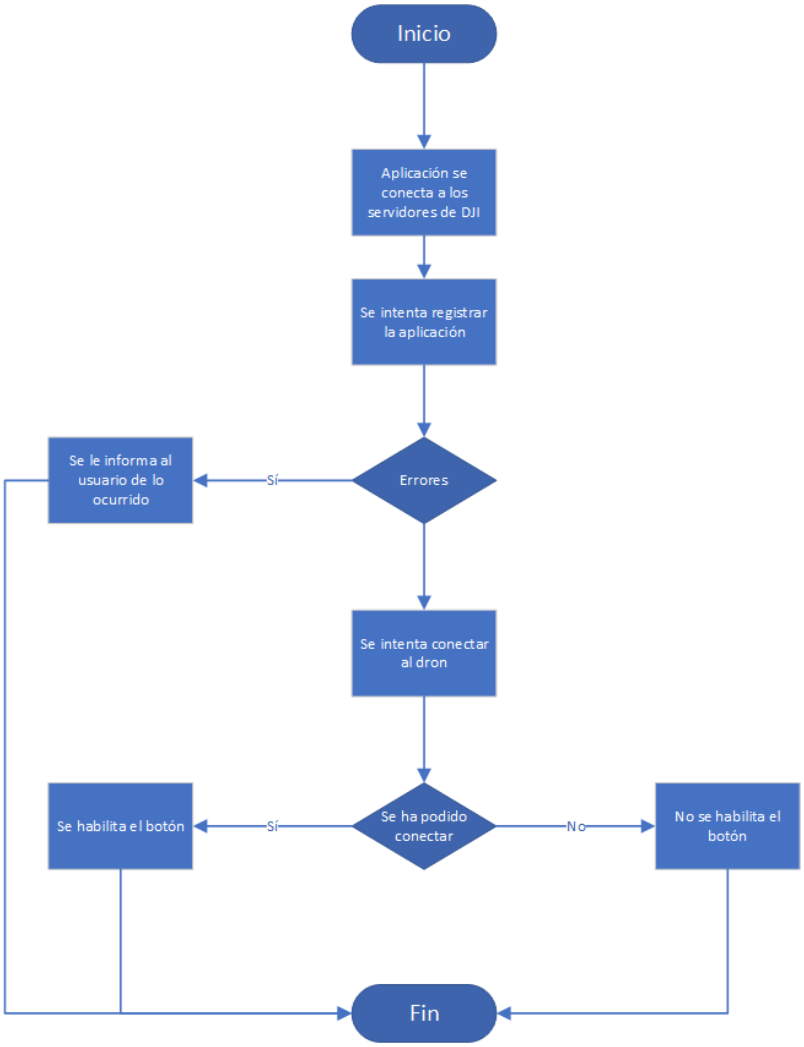


Ilustración 60. Diagrama de flujo Connection Activity

2.6.1.2. Create Mission

Una vez el usuario ha alcanzado esta "Activity" la aplicación comprueba, en primer lugar, si tiene algún registro de la última URL válida introducida, en el caso afirmativo escribe automáticamente dicho registro en el campo correspondiente.

La aplicación en ese momento espera hasta que el usuario pulse alguno de los dos botones de la vista. Cuando ello ocurre, obtiene el texto introducido en el campo referente a la URL del servidor. En función de si ha pulsado sobre el botón de crear una nueva misión o sobre el de acceder a una misión, se realizan procedimientos diferentes, que son los siguientes:

- Botón acceder misión: se obtiene el ID de la misión introducido por el usuario. Se genera una petición API Rest al servidor y se le envía dicho ID. El servidor proporciona una respuesta en función del ID proporcionado. Si el resultado es correcto, se almacena dicho ID para poder transmitirlo a futuras "Activities".

En cuanto a la implementación realizada para este procedimiento tenemos la siguiente:

```
private void checkIDmission()
{
    if(!getUrlString().equals("") && !getIDmission().equals(""))
    {
        // Instantiate the RequestQueue.
        RequestQueue queue = Volley.newRequestQueue( context: this);
        String url = HTTP + getUrlString() + "/checkMissionID/" + getIDmission();
        // Request a string response from the provided URL.
        StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
                new Response.Listener<String>()
                {
                    @Override
                    public void onResponse(String response)
                    {
                        runOnUiThread(new Runnable()
                        {
                            @Override
                            public void run()
                            {
                                savePreferencies();

                                Intent intent = new Intent(getApplicationContext(),
                                        MainActivity.class);
                                intent.putExtra( name: "url", getUrlString());
                                intent.putExtra( name: "idMission", getIDmission());
                                intent.putExtra( name: "djiAircraft", getDjiAircraft());
                                startActivity(intent);
                            }
                        });
                    }
                }, new Response.ErrorListener()
        {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                runOnUiThread(new Runnable()
                {
                    @Override
                    public void run()
                    {
                        closeKeyboard();
                        setErrorServerText("La misión " + getIDmission() + " no existe o no " +
                                "se puede" +
                                "contactar con el servidor");
                        errorServerText.setVisibility(View.VISIBLE);
                    }
                });

                Log.d(TAG,  msg: "Respuesta: " + error.toString());
            }
        });

        // Add the request to the RequestQueue.
        queue.add(stringRequest);
    }

    else
        showToast( toastMsg: "Se deben rellenar los dos campos anteriores");
```

Ilustración 61. Create Mission 1

```
            @Override
            public String getBodyContentType() { return "application/json; charset=utf-8"; }
            @Override
            public byte[] getBody() throws AuthFailureError {
                return requestBody == null ? null : requestBody.getBytes(StandardCharsets.UTF_8);
            }
            @Override
            protected Response<String> parseNetworkResponse(NetworkResponse response) {
                String responseString = "";
                if (response != null) {
                    responseString = String.valueOf(response.statusCode);
                    // can get more details such as response.headers
                }
                return Response.success(responseString, HttpHeaderParser
                        .parseCacheHeaders(response));
            }
        };
        requestQueue.add(stringRequest);
    }
    catch (Exception e)
    {
        showToast( toastMsg: "Error al crear una nueva misión");
        Log.d(TAG,  msg: "Salta excepción al crear una nueva misión");
    }
}
```

Ilustración 62. Create Mission 2

Tal y como se puede comprobar en las dos imágenes superiores, primero se comprueba que tanto el campo de la URL del servidor como del ID de la misión se encuentren completados por el usuario. En caso afirmativo se realiza una petición API Rest al servidor de la aplicación. En dicha petición se le pasa por parámetro dentro de la URL el ID de la misión a la que quiere añadirse el usuario de la aplicación.

En función de si existe dicha misión o no, el servidor nos proporcionará resultados totalmente distintos. Si la misión se ha encontrado en la base de datos, nos indicará que sí existe la misión. Entonces el cliente almacenará el contenido de la URL como valor por defecto para la próxima vez que se inicie la aplicación y se cambiará a la vista “MainActivity”.

Por último, la implementación realizada en el servidor para este proceso es la misma que se ha indicado en el párrafo anterior, siendo esta la siguiente:

```
//Método para comprobar que el ID de la misión introducido por el usuario existe
app.get(config.app.base + '/checkMissionID/:IDmission', async (req,res) =>
{
    try
    {
        var aux = await checkIDmission(req.params.IDmission);

        if(aux)
            res.status(200).send({result: 'El ID de la misión existe', error: null});

        else
            res.status(404).send({result: null, error: 'El ID de la misión NO existe'});

    }

    catch(error)
    {
        console.log('Ha habido al comprobar el ID de la misión\n' + `${error}`);
        res.status(404).send({result: null, error: 'Ha habido al comprobar el ID de la misión\n'});
    }
});
```

Ilustración 63. Create Mission 3

- Botón crear misión: se genera una petición API Rest al servidor en la que se le solicita crear una nueva misión. El servidor devuelve una respuesta indicado el resultado de la petición. En el caso de que el resultado sea correcto, el cliente realiza otra petición para obtener el ID de su misión. Una vez recibido se muestra en la pantalla en forma de 'Toast' y procede a pasar a la siguiente "Activity".

  En referencia a la implementación realizada en el cliente para llevar a cabo este proceso de crear una misión tenemos la siguiente:

```
private void createNewMission()
{
    try
    {
        RequestQueue requestQueue = Volley.newRequestQueue( context: this);
        String URL = HTTP + getUrlString() + "/createMission";
        JSONObject jsonBody = new JSONObject();
        final String requestBody = jsonBody.toString();

        StringRequest stringRequest = new StringRequest(Request.Method.POST, URL,
                new Response.Listener<String>() {
            @Override
            public void onResponse(String response)
            {
                //Coordenadas recibidas correctamente por el servidor
                if(response.equals("200"))
                {
                    //Como ha ido bien el registro paso a guardar la url del
                    // servidor introducida por el usuario
                    savePreferencies();
                    getLastMission();
                }
                Log.i( tag: "OnResponse newMission", response);
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                //Saco el mensaje de error
                showToast( toastMsg: "Hay un problema de conexión con el servidor");
                Log.e( tag: "onErrorResponse", error.toString());
            }
        }) {
            @Override
            public String getBodyContentType() { return "application/json; charset=utf-8"; }
            @Override
            public byte[] getBody() throws AuthFailureError {
                return requestBody == null ? null : requestBody.getBytes(StandardCharsets.UTF_8);
            }
            @Override
            protected Response<String> parseNetworkResponse(NetworkResponse response) {
                String responseString = "";
                if (response != null) {
                    responseString = String.valueOf(response.statusCode);
                    // can get more details such as response.headers
                }
                return Response.success(responseString, HttpHeaderParser
                        .parseCacheHeaders(response));
            }
        };
        requestQueue.add(stringRequest);
    }
```

Ilustración 64. Create Mission 4

Tal y como podemos ver en la imagen superior, este método genera una nueva petición API Rest en la que solicita al servidor el crear una nueva misión. Si el servidor contesta con un código 200, que significa que todo ha ido correctamente, el cliente realizará una segunda petición API Rest para conocer el ID de la misión que acaba de generar.

En cuanto a la implementación realizada en el servidor tenemos la siguiente:

```
//Método para crear una nueva misión
app.post(config.app.base + '/createMission', async (req,res) =>
{
    try
    {
        //Primero insertamos una nueva misión en la BD
        await knex('mission').insert({'dronesNumber' : 3, 'distanceBetween' : 1, 'minAltitud' : 20,
        'altitudBetween' : 1, 'missionSpeed' : "Velocidad Media"});

        res.status(200).send({result: "TODO OKAY", error: null});
    }

    catch(error)
    {
        console.log('Ha habido al crear una nueva misión\n' + `${error}`);
        res.status(500).send({result: null, error: 'Ha habido al crear una nueva misión\n'});
    }
});
```

Ilustración 65. Create Mission5

En dicha implementación podemos comprobar cómo se genera una nueva misión por parte del servidor en la base de datos con unos parámetros por defecto. Si no existen problemas se le informa al usuario de ello, mediante el uso de un código HTTP 200. Si llegara a ocurrir algún error, el usuario conocerá de esa contingencia mediante un código 500.

Por último, cuando el cliente solicita el ID de la misión creada, el servidor realiza los siguientes pasos:

```
app.get(config.app.base + '/getLastMission', async (req,res) =>
{
    try
    {
        var aux = await getLastMission();

        if(aux != null)
            res.status(200).send({result: aux, error: null});

        else
            res.status(500).send({result: null, error: 'Ha habido al obtener la última misión creada'});
    }

    catch(error)
    {
        console.log('Ha habido al obtener la última misión creada\n' + `${error}`);
        res.status(404).send({result: null, error: 'Ha habido al obtener la última misión creada\n'});
    }
});
```

Ilustración 66. Create Mission 6

Tal y como se puede ver en la imagen superior, el servidor obtiene de la base de datos la última misión creada por un cliente. Si se obtiene algún ID de misión, éste se le transmite al usuario acompañado de un código 200 y en caso contrario, se le informa al usuario del error utilizando un código 500.

Por último, se procede a mostrar a modo de diagrama de flujo el proceso principal realizado por dicha "Activity":

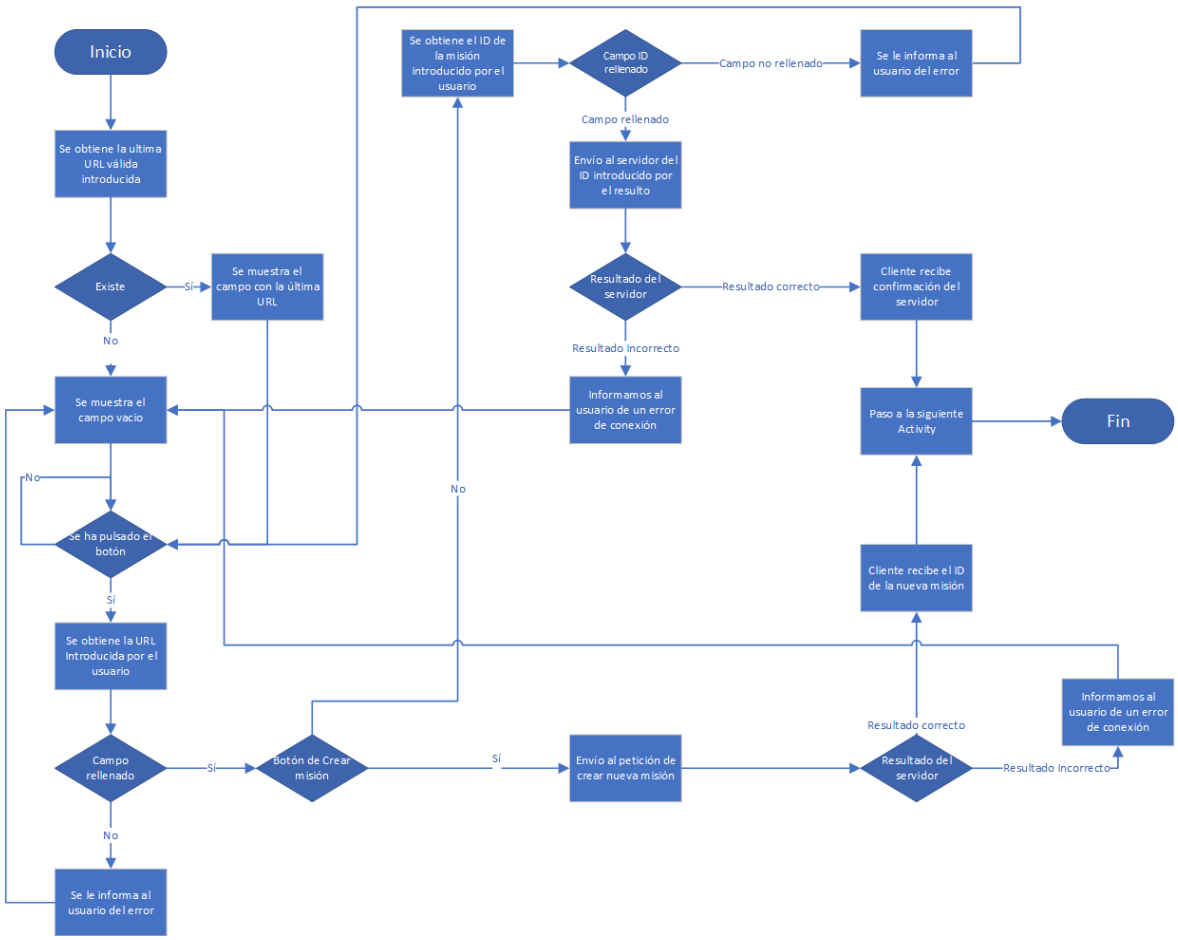


Ilustración 67. Diagrama de flujo Create Mission Activity

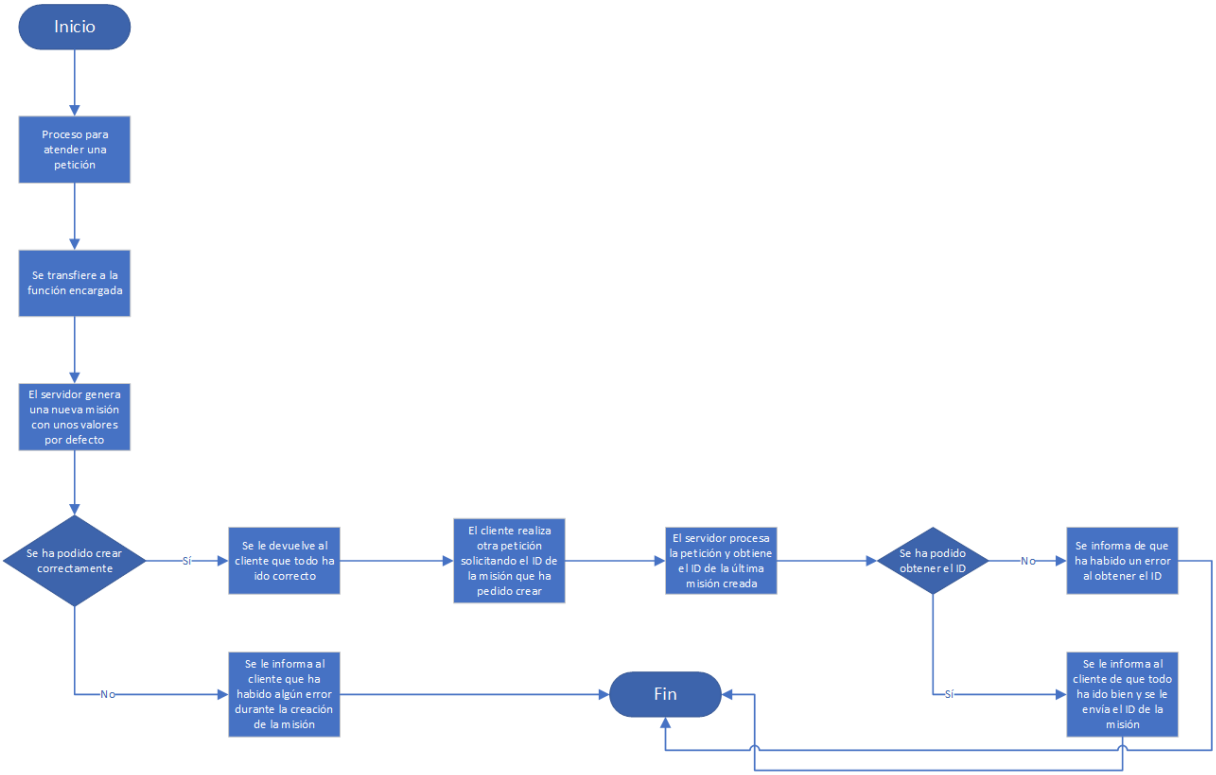


Ilustración 68. Diagrama de flujo Crear Misión Servidor

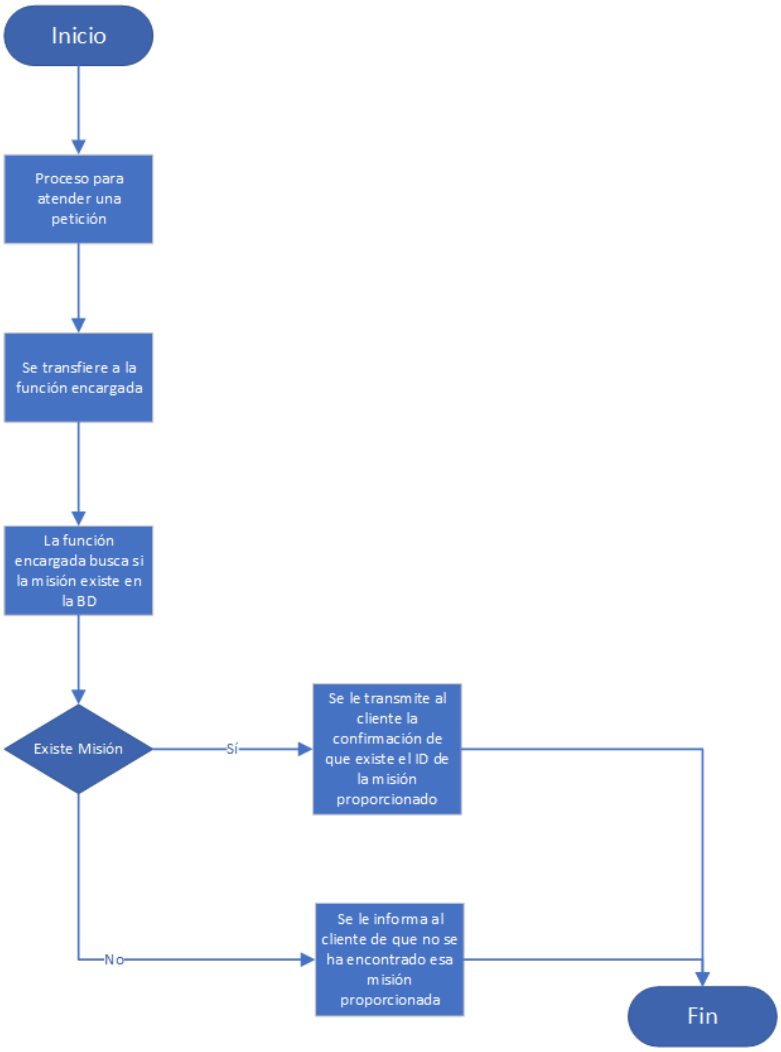


Ilustración 69. Diagrama de flujo Acceder a una Misión Servidor

2.6.1.3. Main Activity

Al acceder a esta "Activity", primero se realiza la obtención del ID de la misión a la que se ha accedido, el modelo del dron conectado y la URL del servidor de la aplicación.

A continuación, la aplicación espera hasta que el usuario pulse el botón de registrar. Una vez lo hace comprueba que se haya seleccionado un tipo de usuario, y es cuando genera una nueva petición API Rest para registrar ese nuevo usuario en la misión.

En función de la respuesta del servidor se le informará al usuario del error ocurrido en el campo destinado para ello o se le dará acceso a la siguiente "Activity".
Una vez explicado el funcionamiento del principal proceso de esta "Activity", podremos analizar la implementación tanto en el cliente como en el servidor.

En el cliente podemos ver que dicha implementación es la que se muestra a continuación:

```
private void registerDrone()
{
    try
    {
        RequestQueue requestQueue = Volley.newRequestQueue( context: this);
        String URL = HTTP + getURL() + "/register";

        JSONObject jsonBody = new JSONObject();
        jsonBody.put( name: "userType", getUserTypeString());
        jsonBody.put( name: "idMission", getIdMission());
        jsonBody.put( name: "djiAircraft", getDjiAircraft());
        final String requestBody = jsonBody.toString();

        StringRequest stringRequest = new StringRequest(Request.Method.POST, URL,
                new Response.Listener<String>() {
            @Override
            public void onResponse(String response)
            {
                //No hay un usuario administrador registrado
                if(response.equals("200"))
                {
                    runOnUiThread(new Runnable()
                    {
                        @Override
                        public void run()
                        {
                            //Saco el mensaje de error
                            setDronesRegisterString("Se debe registrar en primer lugar " +
                                    "un usuario administrador");
                            errorRegisterView.setVisibility(View.VISIBLE);
                        }

                    });
                }

                //Drone registrado correctamente
                else if(response.equals("201"))
                    getDroneID();

                //Caso en el que no se admiten más drones
                else if(response.equals("202"))
                {
                    runOnUiThread(new Runnable()
                    {
                        @Override
                        public void run()
                        {
                            //Saco el mensaje de error
                            setDronesRegisterString("Pide al administrador que amplie" +
                                    " el número de drones admitidos");
                            errorRegisterView.setVisibility(View.VISIBLE);
                        }
```

Ilustración 70. Main Activity 1

```
            //Caso en el que se intenta registrar un segundo administrador
            else if(response.equals("203"))
            {
                runOnUiThread(new Runnable()
                {
                    @Override
                    public void run()
                    {
                        //Saco el mensaje de error
                        setDronesRegisterString("Sólo se puede registrar a un " +
                                "administrador por misión");
                        errorRegisterView.setVisibility(View.VISIBLE);
                    }

                });
            }

            Log.i( tag: "OnResponse", response);
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error)
        {
            //Saco el mensaje de error
            setDronesRegisterString("Hay un problema de conexión con el servidor");
            errorRegisterView.setVisibility(View.VISIBLE);
            Log.e( tag: "onErrorResponse", error.toString());
        }
    }) {
        @Override
        public String getBodyContentType() { return "application/json; charset=utf-8"; }

        @Override
        public byte[] getBody() throws AuthFailureError {
            return requestBody == null ? null : requestBody.getBytes(StandardCharsets.UTF_8)
        }

        @Override
        protected Response<String> parseNetworkResponse(NetworkResponse response) {
            String responseString = "";
            if (response != null) {
                responseString = String.valueOf(response.statusCode);
                // can get more details such as response.headers
            }
            return Response.success(responseString, HttpHeaderParser
```

Ilustración 71. Main Activity 2

```
@Override
public void onClick(View view)
{
    switch (view.getId())
    {
        case R.id.registerButton:
        {
            if(checkFields())
            {
                //Registramos a un usuario
                registerDrone();
            }

            else
                showToast( toastMsg: "Se debe indicar la URL del servidor y el tipo de usuario");
        }
    }
}
```

Ilustración 72. Main Activity 3

Tal y como se puede comprobar utilizando las tres imágenes superiores. Al pulsar el botón registrar se procederá a comprobar que se haya introducido el tipo de usuario a registrar y que la URL se haya obtenido correctamente desde la "Activity" anterior.

En el caso que tanto la URL como el tipo de usuario tengan valores correctos se procederá a registrar al nuevo usuario y, en caso contrario, se le informará del error ocurrido.

Una vez se ha conseguido acceder al método "registerDrone", el cliente preparará una nueva petición API Rest, e introducirá en el cuerpo del mensaje un objeto de tipo JSON para transmitir al servidor el tipo de usuario a registrar, el ID de la misión donde se debe registrar ese usuario, así como el modelo del dron asociado.

Una vez realizada dicha petición el servidor nos ofrecerá una respuesta en función de los datos proporcionados por la aplicación. En cuanto a la implementación realizada en el servidor podemos comprobar que es la siguiente:

```
//Método para realizar el registro de un nuevo drone
app.post(config.app.base + '/register', async (req,res) =>
{
    try
    {
        //Primero deberemos de comprobar que el drone puede incluirse en la misión
        var maxDronesMission = await getMaxDroneMission(req.body.idMission);
        var dronesRegistered = await getDronesRegistered(req.body.idMission);
        var adminUser = await getAdminUserNumber(req.body.idMission);


        //Insertamos en la tabla de los drones el nuevo drone
        if(dronesRegistered < maxDronesMission)
        {
            //Tengo que comprobar que tengo primero un drone con un usuario administrador
            if(req.body.userType == "Usuario" && adminUser == 0)
                res.status(200).send({result: null, error: 'Se debe registrar en primer lugar un usuario administrador'});

            //Sólo puedo registrar a un usuario administrador
            else if(adminUser == 1 && req.body.userType == "Usuario")
            {
                await knex('drones').insert({'userType' : req.body.userType, 'idMission' : req.body.idMission, 'djiModel' : req.body.djiAircraft});
                await updateDronesAltitud(req.body.idMission);
                res.status(201).send({result: 'Drone registrado correctamente', error: null});
            }

            else if(adminUser == 0 && req.body.userType == "Administrador")
            {
                await knex('drones').insert({'userType' : req.body.userType, 'idMission' : req.body.idMission, 'djiModel' : req.body.djiAircraft});
                await updateDronesAltitud(req.body.idMission);
                res.status(201).send({result: 'Drone Administrador registrado correctamente', error: null});
            }

            else
                res.status(203).send({result: null, error: 'Sólo se puede registrar a un administrador por misión'});
        }

        //Si no caben más drones
        else
            res.status(202).send({result: null, error: 'Pide al administrador que amplie el número de drones admitidos'});

    }

    catch(error)
    {
        console.log('Ha habido un error al registrar.\n' + `${error}`);
        res.status(400).send({result: null, error: 'Ha habido un error al registrar.'});
    }
});
```

Ilustración 73. Main Activity 4

Tal y como se puede ver, en primer lugar, se obtiene el número de drones que se pueden registrar en la misión proporcionada, el número de drones registrados actualmente y si existe algún usuario de tipo Administrador en ese momento.

Con todos los datos necesarios obtenidos de la base de datos, ya podemos comprobar si existe suficiente espacio en la misión para registrar a un nuevo dron. En caso afirmativo, se comprueba si existe algún usuario de tipo Administrador registrado. En caso de no existir ninguno, éste debe ser Administrador, ya que de acuerdo a la política de la aplicación el primer usuario de una misión debe ser de este tipo. Si se satisfacen todas las condiciones, se inserta el usuario en la base de datos y se le notifica al cliente del resultado.

Si, por el contrario, no se ha podido cumplir ambas condiciones, se comprueba que el usuario a registrar no es Administrador y que ya existe uno de este tipo previamente. En caso afirmativo se inserta en la base de datos y se le notifica al cliente de la acción ocurrida.

Por ello, tal y como se puede comprobar en el siguiente diagrama de flujo, en función de los parámetros proporcionados por el cliente, el servidor realizará diferentes comprobaciones, de las que devolverá un resultado. Posteriormente, el cliente utilizará esa información transmitida por el servidor para informar al usuario de la aplicación de algún error o en caso contrario, concederle acceso a la siguiente "Activity", la cual dependerá del tipo de usuario registrado.

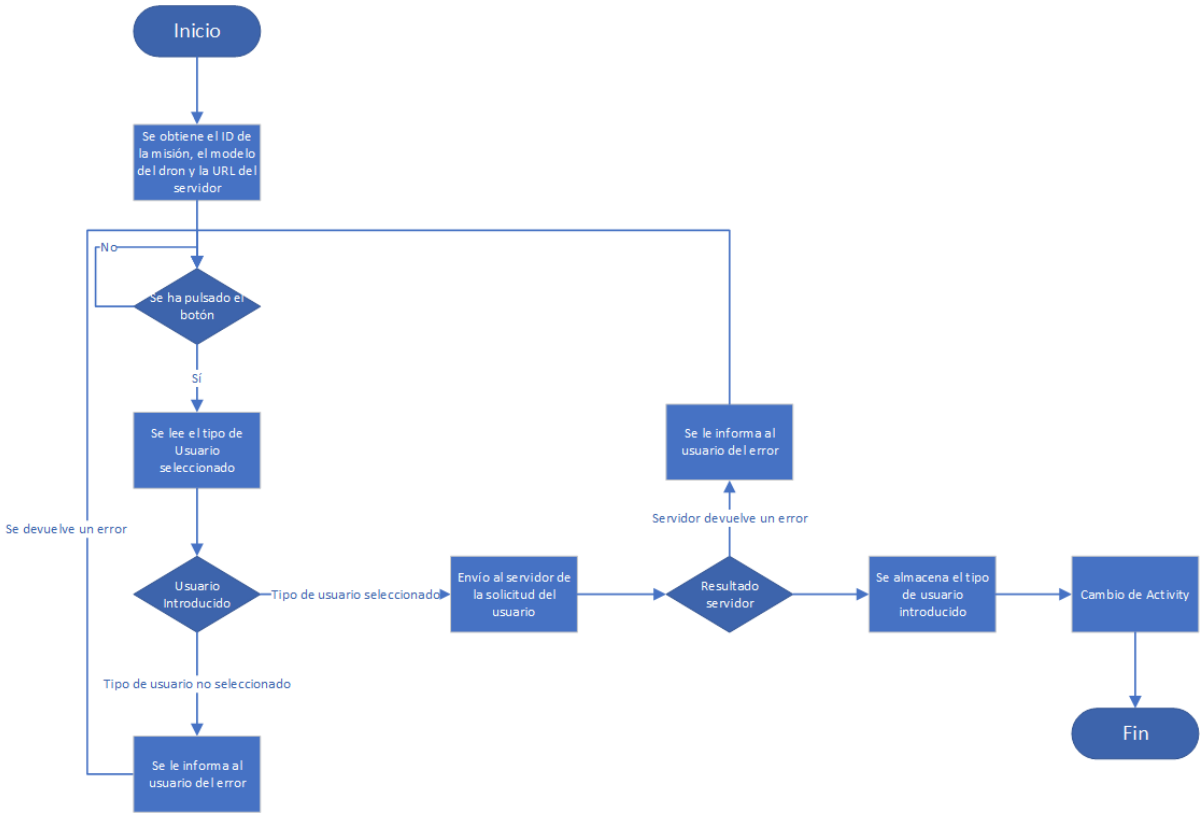


Ilustración 74. Diagrama de flujo Main Activity

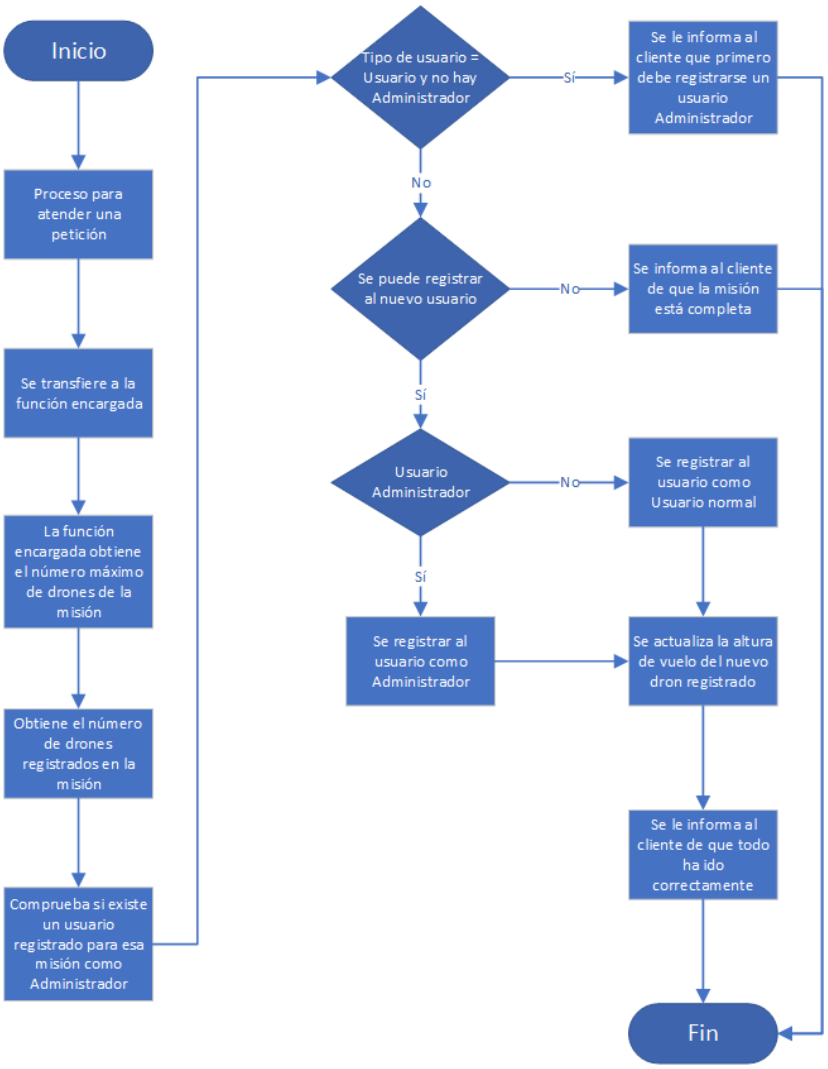


Ilustración 75. Diagrama de flujo Main Activity Servidor

2.6.1.4. Fly Registry

Durante esta "Activity" los usuarios Administradores podrán modificar los parámetros de la misión, ya sea porque ésta contiene los valores por defecto después de crearla o bien, porque se desean modificar los existentes.
Por ello, una vez la aplicación haya detectado que el usuario ha pulsado el botón de "Actualizar", ésta comprobará que el usuario Administrador haya introducido una opción válida en cada uno de los desplegables, así como se haya dibujado la formación a realizar por los drones utilizados para esa misión.

Una vez comprobado lo anterior, la aplicación cliente volverá a generar una nueva petición HTTP, en la que incluirá como cuerpo de la misma, un objeto JSON con el número de drones, distancia entre los vehículos, altura mínima de vuelo, la altura entre drones y el ID de la misión sobre la que se tienen que realizar dichos cambios.

Una vez enviada la petición al servidor, si los datos referentes a la misión se han podido actualizar correctamente, se le informará al usuario de ello y se cambiará a la vista para los usuarios Administradores.

En cuanto a la implementación para llevar a cabo los procesos explicados es la siguiente:

```
private void updateMissionData()
{
    try
    {
        RequestQueue requestQueue = Volley.newRequestQueue( context: this);
        String URL = HTTP + getURL() + "/updateMissionData";

        JSONObject jsonBody = new JSONObject();
        jsonBody.put( name: "dronesNumber", getDronesNumber());
        jsonBody.put( name: "dronesDistance", getDronesDistance());
        jsonBody.put( name: "minumunAltitude", getMinumunAltitude());
        jsonBody.put( name: "altitudeBetweenDrones", getAltitudeBetweenDrones());
        jsonBody.put( name: "idMission", getIdMission());
        final String requestBody = jsonBody.toString();

        StringRequest stringRequest = new StringRequest(Request.Method.PUT, URL,
                new Response.Listener<String>() {
            @Override
            public void onResponse(String response)
            {
                //No hay un usuario administrador registrado
                if(response.equals("200"))
                {
                    runOnUiThread(new Runnable()
                    {
                        @Override
                        public void run()
                        {
                            showToast( toastMsg: "Misión actualizada correctamente");

                            Intent intent = new Intent(getApplicationContext(),
                                    MissionActivity.class);
                            intent.putExtra( name: "userType", getUserType());
                            intent.putExtra( name: "url", getURL());
                            intent.putExtra( name: "idMission", getIdMission());
                            intent.putExtra( name: "djiAircraft", getDjiAircraft());
                            intent.putExtra( name: "idDrone", getIdDrone());
                            startActivity(intent);
                        }
                    });
                }
                Log.i( tag: "OnResponse", response);
```

Ilustración 76. Fly Registry 1

```
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                //Saco el mensaje de error
                showToast( toastMsg: "Hay un problema de conexión con el servidor");
                Log.e( tag: "onErrorResponse", error.toString());
            }
        }) {
            @Override
            public String getBodyContentType() { return "application/json; charset=utf-8"; }

            @Override
            public byte[] getBody() throws AuthFailureError {
                return requestBody == null ? null : requestBody.getBytes(StandardCharsets.UTF_8);
            }

            @Override
            protected Response<String> parseNetworkResponse(NetworkResponse response) {
                String responseString = "";
                if (response != null) {
                    responseString = String.valueOf(response.statusCode);
                    // can get more details such as response.headers
                }
                return Response.success(responseString, HttpHeaderParser
                        .parseCacheHeaders(response));
            }
        };

        requestQueue.add(stringRequest);
    }

    catch (Exception e)
    {
        showToast( toastMsg: "Error al actualizar los datos de la misión");
        Log.d(TAG,  msg: "Salta excepción al actualizar los datos de la misión");
    }
}

/**
```

Ilustración 77. Fly Registry 2

Además, tal y como se puede comprobar en la primera imagen, en caso de no recibir un código 200 por parte del servidor, no se le concederá al usuario acceso a la siguiente "Activity" y se le informará del error ocurrido.

En cuanto a la implementación llevada a cabo en el servidor tenemos la siguiente:

```
//Método para actualizar los datos de la misión
app.put(config.app.base + '/updateMissionData', async (req,res) =>
{
    try
    {
        var aux = await updateMissionData(req.body.idMission, req.body.dronesNumber, req.body.dronesDistance,
            req.body.minumunAltitude, req.body.altitudeBetweenDrones);

        if(aux)
        {
            await updateDronesAltitud(req.body.idMission);
            res.status(200).send({result: 'Misión actualizada correctamente', error: null});
        }


        else
            res.status(400).send({result: null, error: 'No se ha podido actualizar la misión en la DB.'});
    }

    catch(error)
    {
        console.log('Ha habido al actualizar la misión.\n' + `${error}`);
        res.status(400).send({result: null, error: 'Ha habido al actualizar la misión.'});
    }
});
```

Ilustración 78. Fly Registry 3

Tal y como se puede comprobar en la imagen superior, el servidor procede a actualizar los datos de la misión proporcionados por el cliente. Si no se produce ningún problema, el servidor procederá a modificar la altura de vuelo de cada uno de los drones registrados en esa misión. Esta acción se realiza debido a que, como hemos indicado en anteriores puntos de la memoria, por motivos de seguridad se va a mantener una altura entre cada uno de los drones, que es indicada por el usuario. Por ello, una vez actualizados los datos de la misión, el servidor procederá a modificar la altura de vuelo de cada uno de los drones registrados.

Este proceso de actualización de la altura de vuelo se lleva a cabo también al registrar un nuevo dron en una misión. De esta manera se consigue que cada vehículo mantenga una altura de seguridad.

Por último, y al igual que se ha realizado con las “Activities” anteriores, procederemos a mostrar los diagramas de flujo empleados tanto para el cliente como para el servidor.

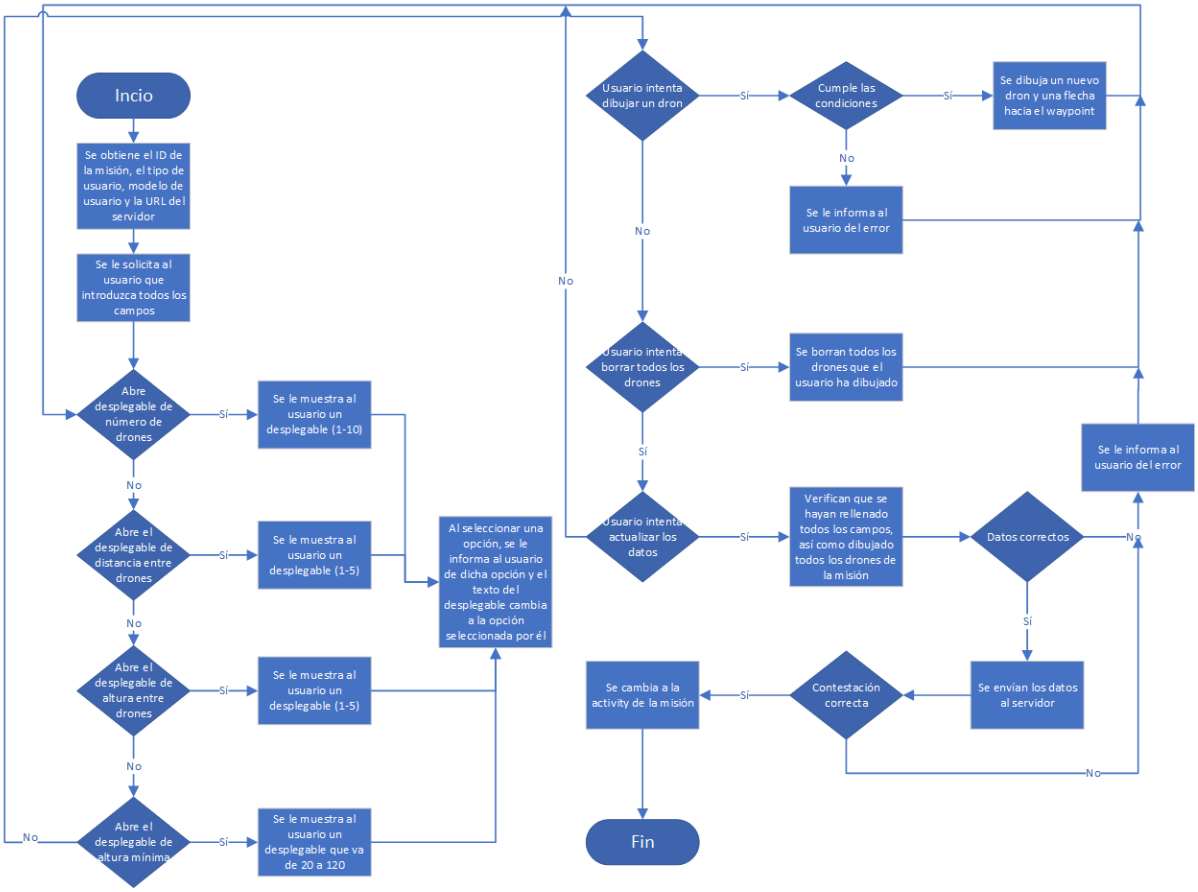


Ilustración 79. Diagrama de flujo Fly Registry Activity

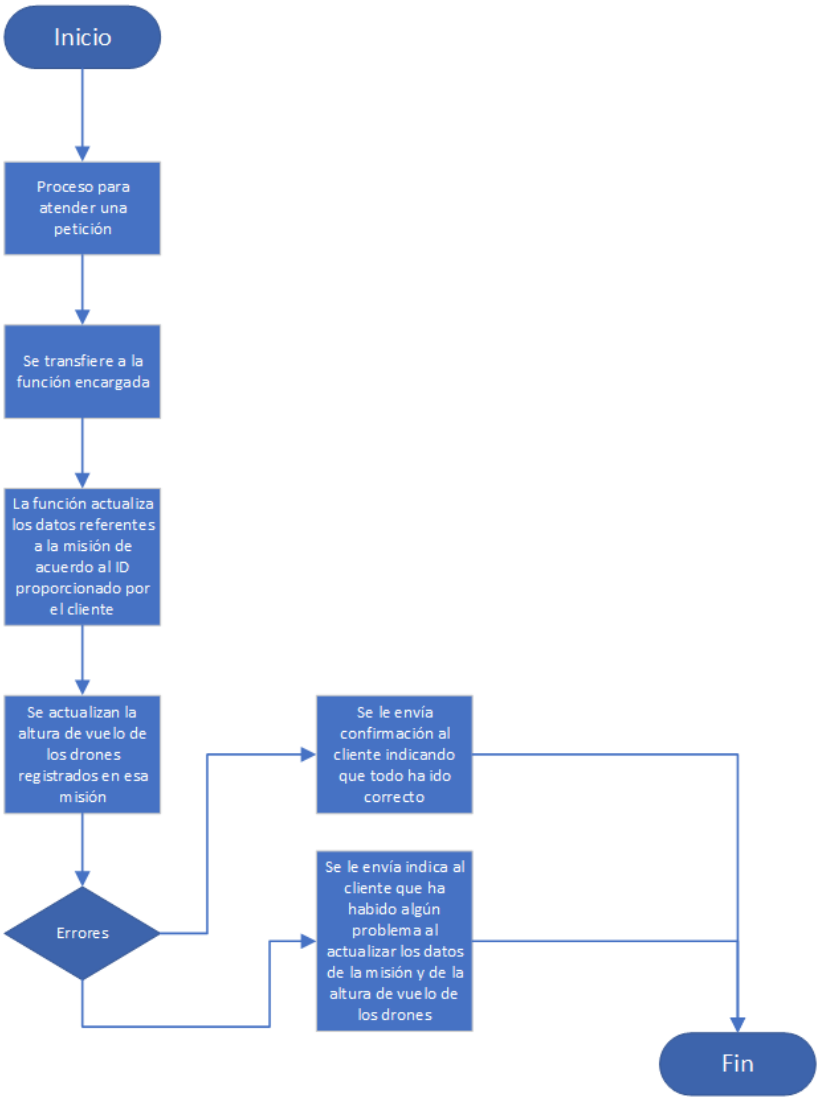


Ilustración 80. Diagrama de flujo Fly Registry Servidor

2.6.1.5. Mission Activity

Esta “Activity” tal y como se ha venido comentando a lo largo de este trabajo es la que dispone de la mayor cantidad de utilidades y, por lo tanto, de opciones independientemente de si el usuario registrado es Administrador o no.

La “Activity” dispone de las funcionalidades explicadas en apartados previos de esta memoria, y ahora se procederá a analizar la implementación realizada para cada una de las funciones utilizadas, tanto del servidor como del cliente. Destacar que, al igual que se ha venido realizando a lo largo de este apartado, se incluirá al final de la presente explicación unos diagramas de flujo, que ayudarán a entender el funcionamiento de esta “Activity”.

A continuación, se presentan dos imágenes en las que se indican cada una de las funciones ejecutadas por la aplicación en función, de la acción realizada por el usuario. Por ello, a lo largo de esta explicación procederemos a analizar la implementación realizada para cada una de ellas.

```
@Override
public void onClick(View v)
{
    switch (v.getId())
    {
        case R.id.backButton:
        {
            finish();
            break;
        }

        case R.id.missionButton:
        {
            //Abro el popup
            openPopupMission();
            break;
        }

        case R.id.droneCameraButton:
        {
            Intent uxView = new Intent( packageContext: this, UXactivity.class);
            startActivity(uxView);
            break;
        }

        case R.id.addWaypointsButton:
        {
            enableDisableAdd();
            break;
        }

        case R.id.clearWaypointsButton:
        {
            this.mMap.clear();
            this.sendWaypoints.clear();
            this.receivedWaypoints.clear();

            waypointList.clear();
            //waypointMissionBuilder.waypointList(waypointList);
            updateDroneLocation();
            break;
        }

        case R.id.waypointsButton:
        {
            getServerWaypoints();

            break;
```

Ilustración 81. Mission Activity 1

```
case R.id.mapType:
{
    getMapChargeView();
    break;
}

//Botones del layout de la misión
case R.id.sendWaypointsButton:
{
    sendWaypoints();
    break;
}

case R.id.geoButton:
{
    Intent intent = new Intent(getApplicationContext(), GEO_Activity.class);
    startActivity(intent);

    break;
}

case R.id.startButton:
{
    sendStartMission();
    break;
}

case R.id.stopButton:
{
    sendStopMission();
    break;
}

case R.id.pauseButton:
{
    sendResumePauseMission();
    break;
}

//Botones del layout de seleccionar el mapa
case R.id.mapButton:
{
    setMapMode();
    break;
}

case R.id.satelliteButton:
{
    setSatelliteMode();
    break;
}

case R.id.reliefButton:
{
    setReliefMode();
    break;
}
```

Ilustración 82. Mission Activity 2

Respecto a las opciones relacionadas con el control de la misión, debemos recordar que dichas funciones, tan sólo se encontrarán disponibles para usuarios Administradores. En el caso de que el Administrador presionara el botón "Misión", tal y como se indica en la imagen superior se ejecutaría el método "openMissionPopup" que está implementado de la siguiente manera:

```
//Método para abrir e inicializar el popup del botón de misión
private void openPopupMission()
{
    RelativeLayout relativeLayout = (RelativeLayout) getLayoutInflater()
            .inflate(R.layout.mission_popup, root: null);
    Button pauseButton = relativeLayout.findViewById(R.id.pauseButton);
    Button startButton = relativeLayout.findViewById(R.id.startButton);
    Button stopButton = relativeLayout.findViewById(R.id.stopButton);
    Button geoButton = relativeLayout.findViewById(R.id.geoButton);
    this.spinnerSpeed = relativeLayout.findViewById(R.id.speedSpinner);
    Button sendWaypoints = relativeLayout.findViewById(R.id.sendWaypointsButton);

    pauseButton.setOnClickListener(this);
    startButton.setOnClickListener(this);
    stopButton.setOnClickListener(this);
    geoButton.setOnClickListener(this);
    sendWaypoints.setOnClickListener(this);

    //Ahora inicializamos el spinner
    initSpinnerMission(spinnerSpeed);

    AlertDialog.Builder alertDialog = new AlertDialog.Builder( context: this);
    alertDialog.setTitle("");
    alertDialog.setView(relativeLayout);
    alertDialog.create().show();
}
```

Ilustración 83. Mission Activity 3

En dicho método cargamos el "layout" llamado "mission_popup". Una vez cargado obtenemos una referencia a cada uno de los elementos gráficos que se incluyen en él. Además, añadimos un evento sobre cada uno de los botones para poder ejecutar ciertos métodos una vez el usuario presione sobre ellos.

En el caso de presionar el botón de pausa, la aplicación ejecutará el método "sendResumePauseMission". Éste nos permitirá reanudar o pausar una misión. En cuanto a la implementación de este método tenemos la siguiente:

```java
//Método para mandar el mensaje de que se desea reanudar o pausar la misión
private void sendResumePauseMission()
{
    try
    {
        RequestQueue requestQueue = Volley.newRequestQueue( context: this);
        String URL = HTTP + getURL() + "/sendPauseResumeMissionNotification";
        JSONObject jsonBody = new JSONObject();
        jsonBody.put( name: "idMission", getIdMission());

        final String requestBody = jsonBody.toString();

        StringRequest stringRequest = new StringRequest(Request.Method.POST, URL,
                new Response.Listener<String>() {
            @Override
            public void onResponse(String response)
            {
                Log.i( tag: "OnResponse PauseResume", response);
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                //Saco el mensaje de error
                showToast( toastMsg: "Hay un problema de conexión con el servidor");
                Log.e( tag: "onErrorResponse", error.toString());
            }
        }) {
            @Override
            public String getBodyContentType() { return "application/json; charset=utf-8"; }

            @Override
            public byte[] getBody() throws AuthFailureError {
                return requestBody == null ? null : requestBody.getBytes(StandardCharsets.UTF_8);
            }

            @Override
            protected Response<String> parseNetworkResponse(NetworkResponse response) {
                String responseString = "";
                if (response != null) {
                    responseString = String.valueOf(response.statusCode);
                    // can get more details such as response.headers
                }
                return Response.success(responseString, HttpHeaderParser.
                        parseCacheHeaders(response));
            }
        };

        requestQueue.add(stringRequest);
    }
```

Ilustración 84. Mission Activity 4

Tal y como se ha comentado, previamente, este método sirve para notificar al resto de clientes Android que la misión se debe de pausar o reanudar. Pero debemos tener en cuenta que, debido a la necesidad de notificación a los clientes registrados en una misión desde un terminal móvil, ha sido necesario el uso de Firebase de Google. Con esta herramienta se ha procedido a notificar a cada uno de los terminales de cualquier acción a llevar a cabo por el dron.

Por ello, este método lo que hace es enviar un mensaje HTTP al servidor para que éste a su vez, a través de Firebase remita una notificación a todos los clientes de la misión.

En cuanto a la implementación realizada en el servidor para llevar a cabo este tipo de acciones tenemos la siguiente:

```
//Método para pausar o reanudar una misión
app.post(config.app.base + '/sendPauseResumeMissionNotification', async (req,res) =>
{
    try
    {
        var tokensMission = await getTokensMission(req.body.idMission);

        if(tokensMission != null)
        {
            const data = {
                mission: req.body.idMission,
                title: "Reanudar/Pausar Misión",
                message: "Reanudar/Pausar la misión: " + req.body.idMission,
                tokens: tokensMission
            }

            notification.sendPushToMission(data);
            res.status(200).send("Sending Resume/Pause Notification...");
        }

        else
        {
            console.log('Ha habido un error al obtener los tokens\n' + `${error}`);
            res.status(500).send({result: null, error: 'Ha habido un error al obtener los tokens\n'});
        }

    }

    catch(error)
    {
        console.log('Ha habido un error al mandar las notificaciones de detener misión\n' + `${error}`);
        res.status(500).send({result: null, error: 'Ha habido error un al mandar las notificaciones de detener misión\n'});
    }
});
```

Ilustración 85. Mission Activity 5

Tal y como se puede apreciar en la imagen superior, el servidor recabará todos los 'tokens' registrados de cada uno de los usuarios asociados a la misión proporcionada por el cliente. Una vez obtenidos dichos 'tokens', se procederá a generar un objeto JSON para especificar los parámetros del mensaje a enviar mediante Firebase. Cuando éste se encuentra preparado para su envío se le pasa a una función llamada "sendPushToMission", que se encuentra en otro fichero distinto al del

servidor. Ese fichero adicional está siendo utilizado exclusivamente para enviar mensajes mediante Firebase.

Una vez el método implementado en el fichero auxiliar recibe el mensaje, se lo pasa a la función "sendMessage" para su envío a los servidores de Google y posterior recepción en los clientes de la aplicación.

```
function sendPushToMission(notification)
{
    const message = {
        data: {
            title: notification.title,
            message: notification.message,
            mission: notification.mission
        },
        tokens: notification.tokens,
    }

    sendMessage(message);
}

module.exports = { sendPushToOneUser, sendPushToMission }

function sendMessage(message)
{
    admin.messaging().sendMulticast(message)
        .then((response) =>
        {
            console.log("Successfully sent message: ", response);
        })
        .catch((error) =>
        {
            console.log("Error sending message: ", error);
        });
}
```

Ilustración 86. Mission Activity 6

Cabe destacar que el proceso para iniciar una misión o detenerla es exactamente el mismo. Con la única salvedad, en cuanto al contenido del mensaje que se envía, ya que será la aplicación cliente la que determine en función del mensaje recibido, la acción a realizar.

En cuanto a la implementación en la aplicación Android para atender a las notificaciones que ésta recibe tenemos el siguiente código:

```
@Override
public void onMessageReceived(@Nullable RemoteMessage remoteMessage)
{
    super.onMessageReceived(remoteMessage);
    Log.d(TAG,  msg: "Mensaje Recibido");

    String title = "";
    String message = "";

    Map<String, String> data = remoteMessage.getData();

    if(data.size() > 0)
    {
        title = data.get("title");
        message = data.get("message");
    }

    else
    {
        RemoteMessage.Notification notification = remoteMessage.getNotification();
        title = notification.getTitle();
        message = notification.getBody();
    }

    sendNotification(title, message);

    //Enviamos la señal de iniciar la misión
    sendStartMissionOrder(title);
}
```

Ilustración 87. Mission Activity 7

Cabe destacar que dicha clase se encuentra definida en el fichero “AndroidManifest.xml” como un servicio. Lo que se consigue al declarar una clase en el archivo ‘Manifest’ de esta manera, es que dicha clase siempre se esté ejecutando en segundo plano a la espera de mensajes para la aplicación.

En cuanto al método “sendNotification” sirve para sacar de una manera visual el mensaje recibido como una notificación cualquiera de una aplicación, es decir, produce un resultado similar a la que se obtiene al recibir un mensaje en la aplicación “Whatsapp”. La función que sí que se encarga de transferir el mensaje a la “Activity” de la misión es la función “sendStartMissionOrder”, que se encuentra implementada de la siguiente manera:

```
//Método para mandar un mensaje al MissionActivity
private void sendStartMissionOrder(String title)
{
    Intent intent = new Intent(SERVICE_MESSAGE);
    intent.putExtra(MISSION_KEY, title);

    LocalBroadcastManager.getInstance(getApplicationContext())
            .sendBroadcast(intent);
}
```

Ilustración 88. Mission Activity 8

Además, para atender el envío del mensaje a la "Activity" de la misión por parte de esta clase tenemos el siguiente código, que nos permite ejecutar diversas acciones sobre la misión del dron en función del contenido del mensaje.

```
protected BroadcastReceiver mMissionReceiver = (context, intent) -> {
        String message = intent.getStringExtra(MyFirebaseMessagingService.MISSION_KEY);

        if(message.equals("Inicio Misión"))
        {
            if(receivedWaypoints.isEmpty())
                showToast( toastMsg: "Debes obtener primero los waypoints");

            //Le paso al drone los puntos que tiene que hacer
            else
                setWaypointsMission();

            if(receivedWaypoints.isEmpty() || waypointList.isEmpty())
                showToast( toastMsg: "No hay ningún waypoint añadido");

            else
            {
                try
                {
                    //Actualizamos la posición del drone
                    updateDroneLocation();

                    //Configuramos la misión
                    configWayPointMission();

                    //Actualizamos los parámetros de los waypoints
                    uploadWayPointMission();

                    //Ejecutamos la misión
                    startWaypointMission();
                }
```

Ilustración 89. Mission Activity 9

```
                catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }

        else if(message.equals("Detener Misión"))
            stopWaypointMission();

        else if(message.equals("Reanudar/Pausar Misión"))
        {
            if(getWaypointMissionOperator().getCurrentState()
                    == WaypointMissionState.EXECUTION_PAUSED)
                resumeWaypointMission();

            else if(getWaypointMissionOperator().getCurrentState()
                    == WaypointMissionState.EXECUTING)
                pauseWaypointMission();
        }
```

Ilustración 90. Mission Activity 10

En referencia al botón de "GEO SYSTEM", que también aparece una vez cargamos la ventana de la misión, debemos tener en cuenta que, como ya se ha indicado en apartados previos, dicho botón se utiliza para cambiar a la "Activity" mediante la cual podemos obtener información de las zonas legales para el vuelo de estas pequeñas aeronaves.

La implementación de este paso, debido a que tan sólo consiste en un cambio de "Activity" es la siguiente:

```
case R.id.geoButton:
{
    Intent intent = new Intent(getApplicationContext(), GEO_Activity.class);
    startActivity(intent);

    break;
}
```

Ilustración 91. Mission Activity 11

Por último, y para concluir con la última opción que nos ofrece esta ventana, es la del botón "Enviar Waypoints". A través esta opción podremos enviar los "Waypoints" introducidos por el usuario al servidor para que éste calcule las coordenadas relativas a cada uno de los puntos para que los drones puedan seguir manteniendo la formación en todo momento.

Por ello, en cuanto a la implementación de esta funcionalidad se encuentra dentro del método "sendWaypoints".

```
//Método para enviar al servidor los waypoints introducidos por el administrador
private void sendWaypoints()
{
    //Primero compruebo que tengo waypoints
    if(getSendWaypoints().size() == 0)
        showToast( toastMsg: "Debes de introducir al menos un waypoint");

    else if(!getAddWaypointsText().equals("Añadir Waypoints"))
        showToast( toastMsg: "Opción de añadir waypoints marcadas");

    else if(getSpinnerSpeedSelected().equals(velocidad[0]))
        showToast( toastMsg: "Debes seleccionar la velocidad de la misión");

    else
    {
        try
        {
            RequestQueue requestQueue = Volley.newRequestQueue( context: this);
            String URL = HTTP + getURL() + "/sendWaypoints";
            JSONObject jsonBody = new JSONObject();
            jsonBody.put( name: "waypoints", getSendWaypoints());
            jsonBody.put( name: "speed", getSpinnerSpeedSelected());
            jsonBody.put( name: "idMission", getIdMission());

            final String requestBody = jsonBody.toString().replace( target: " Pair", replacement: "")
                    .replace( target: "Pair", replacement: "");

            StringRequest stringRequest = new StringRequest(Request.Method.POST, URL,
                    new Response.Listener<String>() {
                @Override
                public void onResponse(String response)
                {
                    //Coordenadas recibidas correctamente por el servidor
                    if(response.equals("200"))
                    {
                        resetMmap();
                        resetSendWaypoints();
                        showToast( toastMsg: "Waypoints Enviados");
                    }

                    Log.i( tag: "OnResponse sendWaypoint", response);
```

Ilustración 92. Mission Activity 12

```
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                //Saco el mensaje de error
                showToast( toastMsg: "Hay un problema de conexión con el servidor");
                Log.e( tag: "onErrorResponse", error.toString());
            }
        }) {
            @Override
            public String getBodyContentType() { return "application/json; charset=utf-8"; }

            @Override
            public byte[] getBody() throws AuthFailureError {
                return requestBody == null ? null : requestBody
                        .getBytes(StandardCharsets.UTF_8);
            }

            @Override
            protected Response<String> parseNetworkResponse(NetworkResponse response) {
                String responseString = "";
                if (response != null) {
                    responseString = String.valueOf(response.statusCode);
                    // can get more details such as response.headers
                }
                return Response.success(responseString, HttpHeaderParser
                        .parseCacheHeaders(response));
            }
        };

        requestQueue.add(stringRequest);

    }

    catch (Exception e)
    {
        showToast( toastMsg: "Error al enviar los waypoints");
        Log.d(TAG,  msg: "Salta excepción al enviar los waypoints");
    }
}
```

Ilustración 93. Mission Activity 13

Tal y como se puede comprobar en la imagen superior, una vez se empieza a ejecutar este método se comprueba que existen puntos GPS para enviar, que la opción de añadir “Waypoints” se encuentra desactivada y que el usuario Administrador ha introducido la velocidad con la que desea que se lleve a cabo la misión.

Si todo se cumple, la aplicación volverá a generar una nueva petición al servidor para enviarle todos los puntos GPS. En el cuerpo de la petición se incluirá la velocidad de la misión y el ID de la misma, y además se realizará el cálculo los puntos GPS relativos a cada uno de los Waypoints a los que debe dirigirse cada dron de la formación.

Una vez recibida la respuesta del servidor se le informará al usuario del resultado de la petición, es decir, si todo ha ocurrido correctamente o si, por el contrario, ha ocurrido algún error.

En cuanto a la implementación del servidor para realizar este proceso es la siguiente:

```
//Método para realizar el registro de un nuevo drone
app.post(config.app.base + '/sendWaypoints', async (req,res) =>
{
    try
    {
        var aux = req.body.waypoints.split(",");

        //Si no tengo ningún waypoint de misión inserto los que recibo
        if(! await checkMissionWaypoints(req.body.idMission))
        {
            for(var i = 0; i < aux.length; i++)
            {
                aux[i] = aux[i].replace("[{", "").replace("}]", "").replace("{", "").replace("}", "");
                await insertWaypointsMission(req.body.idMission, aux[i]);
            }
        }

        //Sino simplemente los tengo que actualizar
        else if(await checkMissionWaypoints(req.body.idMission))
        {
            //Borro las coordenadas de la misión introducidas
            await knex('coordinates')
                .where({'missionID' : req.body.idMission})
                .del();

            for(var i = 0; i < aux.length; i++)
            {
                aux[i] = aux[i].replace("[{", "").replace("}]", "").replace("{", "").replace("}", "");
                await updateMissionWaypoints(req.body.idMission, aux[i]);
            }
        }

        //Finalmente inserto la velocidad de la misión
        updateSpeedMission(req.body.idMission, req.body.speed);

        //Función para calcular las coordenadas para cada dron
        await calculateRelativeGPSpoints(req.body.idMission);


        res.status(200).send({result: 'Waypoints recibidos correctamente', error: null});
    }

    catch(error)
    {
        console.log('Ha habido al almacenar los waypoints de la misión.\n' + `${error}`);
        res.status(400).send({result: null, error: 'Ha habido al almacenar los waypoints de la misión.\n'});
    }
});
```

Ilustración 94. Mission Activity 14

En la imagen superior puede verificarse que el servidor modifica el 'String' que recibe por parte del cliente. A continuación, el servidor comprueba si existen "Waypoints" registrados en la base de datos para esa misión proporcionada.

En el caso de existir "Waypoints" registrados, se procederá a eliminar todos aquellos que pertenezcan a la misión proporcionada y una vez borrados, se procederá a añadir los nuevos "Waypoints". En el caso de no disponer de ningún registro de puntos GPS para la misión se insertarán en la base de datos las coordenadas GPS enviados por la aplicación cliente.

Finalmente, el método encargado de procesar esta petición HTTP, también se encargará de modificar la velocidad con la que los drones deberán llevar a cabo la misión.

En cuanto a la información del dron, al presionar sobre el botón "Información dron" lo que haremos será cambiar de Activity, tal como se muestra a continuación:

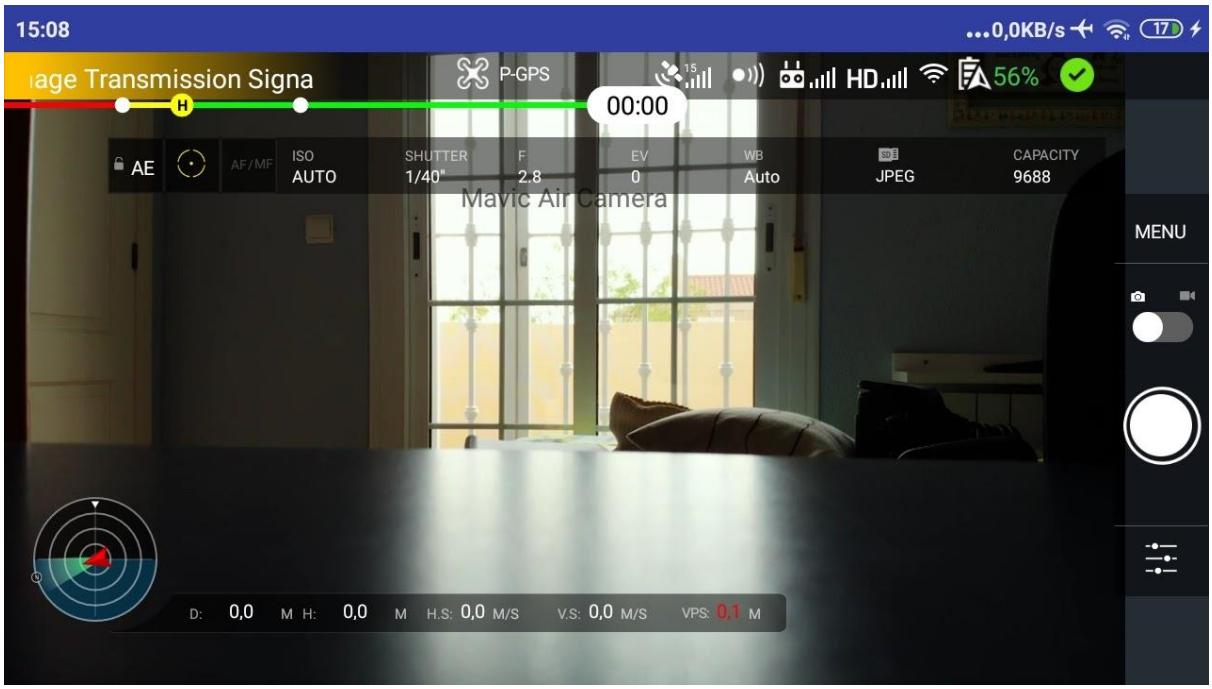


Ilustración 95. Mission Activity 15

Tal y como se puede apreciar en el método "onClick", no finalizamos la "Activity" de la misión, sino que la dejamos en segundo plano. De esta manera, lo que conseguimos es poder seguir ejecutando la misión, al mismo tiempo que podemos ver el estado de la batería, acceder a la cámara del dron, etc.

En cuanto a la opción de añadir "Waypoints" podemos ver que en el método encargado de controlar los eventos relacionados con la pulsación de botones, manda ejecutar el método "enableDisableAdd".

Este último es el que nos permitirá activar o desactivar la función de poder añadir “Waypoints” sobre el mapa, siendo su implementación la que se muestra en la imagen inferior:

```
//Método para habilitar las funciones de administrador
private void enableDisableAdd()
{
    if (getAddWaypointsText().equals("Añadir Waypoints"))
    {
        this.addWaypointsButton.setText("Exit");
        this.clearWaypointsButton.setVisibility(View.VISIBLE);
    }

    else
    {
        this.addWaypointsButton.setText("Añadir Waypoints");
        this.clearWaypointsButton.setVisibility(View.GONE);
    }
}
```

Ilustración 96. Mission Activity 16

Una vez aplicadas las modificaciones realizadas por la función sobre el estado actual de la vista diseño, debemos tener acceso o haber perdido la posibilidad de añadir “Waypoints” sobre el mapa. Si hemos activado la opción de añadir “Waypoints” nos deberá cambiar el texto del botón a ‘Exit’ y aparecerá un nuevo botón para eliminar todos los “Waypoints”, que el usuario haya añadido.

De esta manera, si nos dirigimos al método “onMapClick”, podremos comprobar, que cada vez que el usuario realice un toque sobre el mapa, se le añadirá un marcador sobre éste.

```
@Override
public void onMapClick(LatLng point)
{
    if(getUserType().equals("Administrador"))
    {
        if (getAddWaypointsText().equals("Exit"))
            markWaypoint(point);

        else
            showToast( toastMsg: "Cannot Add Waypoint");
    }
}
```

Ilustración 97. Mission Activity 17

En referencia a la opción de eliminar todos los “Waypoints” del mapa, tan sólo se encontrará disponible cuando tengamos activada la opción añadir “Waypoints”. En cuanto al botón encargado de manejar esta utilidad es el llamado “Eliminar Waypoints”, al presionar sobre él, todos los marcadores del mapa desaparecerán y el listado de “Waypoints” añadidos para ser enviados al

servidor se vaciará tal y como se puede ver en la siguiente implementación realizada dentro del método “onClick”:

```
case R.id.clearWaypointsButton:
{
    this.mMap.clear();
    this.sendWaypoints.clear();
    this.receivedWaypoints.clear();

    waypointList.clear();
    //waypointMissionBuilder.waypointList(waypointList);
    updateDroneLocation();
    break;
}
```

Ilustración 98. Mission Activity 18

Otra opción de las disponibles en esta “Activity” es la descargar los “Waypoints” que debe de ejecutar un dron durante una misión. Para ello, el cliente realizará una nueva petición API Rest y obtendrá dichos puntos GPS, eliminando los iconos preexistentes sobre el mapa, momento en el que se añadirán nuevos marcadores que se situarán sobre las coordenadas recibidas por el servidor. Además, estos marcadores se encontrarán numerados en función del orden con el que se visitarán durante la misión. Los puntos GPS se unirán mediante líneas que simbolizarán la ruta a realizar por el dron. Por último, se actualizará la velocidad a la que debe viajar el vehículo entre “Waypoints” así como, su altitud de vuelo durante la misión.

En cuanto a la implementación realizada sobre el cliente es la descrita en la función “getServerWaypoints”:

```
private void getServerWaypoints()
{
    // Instantiate the RequestQueue.
    RequestQueue queue = Volley.newRequestQueue( context: this);
    String url = HTTP + getURL() + "/getWaypoints/" + getIdMission();

    // Request a string response from the provided URL.
    StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>()
    {
        @Override
        public void onResponse(String response)
        {
            try
            {
                //Vacio el array de markers
                mMarkers.clear();
                mMap.clear();
                receivedWaypoints.clear();
                waypointList.clear();
```

Ilustración 99. Mission Activity 19

```
                JSONObject jsonObject = new JSONObject(response);
                JSONArray jsonArray = jsonObject.getJSONArray( name: "result");

                for(int i = 0; i < jsonArray.length(); i++)
                    addReceivedWaypoint(jsonArray.getJSONObject(i));

                //Ahora obtengo la velocidad con la que tiene que ejecutarse la misión
                getMissionSpeed();

                //Por último obtengo la altura de vuelo
                getFlyAltitud();


            }

            catch (JSONException e)
            {
                e.printStackTrace();
            }
        }

    },

    new Response.ErrorListener()
    {
        @Override
        public void onErrorResponse(VolleyError error) {
            //textView.setText("That didn't work!");
            Log.d(TAG, msg: "onErrorResponse de getServerWaypoints: " + error.toString());
            showToast( toastMsg: "Error al obtener los waypoints");
        }
    });

    // Add the request to the RequestQueue.
    queue.add(stringRequest);
```

Ilustración 100. Mission Activity 20

En cuanto a la implementación realizada en el servidor es la siguiente:

```
//Método para enviar los waypoints que tiene que hacer un drone en particular
app.get(config.app.base + '/getWaypoints/:idMission', async (req,res) =>
{
    try
    {
        var waypoints = await waypointsDrone(req.params.idMission);

        if(waypoints != null)
            res.status(200).send({result: waypoints, error: null});

        else
            res.status(404).send({result: null, error: 'No se han encontrado waypoints para el drone' + req.params.idMission + '.\n'});
    }

    catch(error)
    {
        console.log('Ha habido al recuperar los waypoints del drone ' + req.params.idMission + '.\n' + `${error}`);
        res.status(400).send({result: null, error: 'Ha habido al recuperar los waypoints del drone ' + req.params.idMission + '.\n'});
    }
});
```

Ilustración 101. Mission Activity 21

Tal y como se puede comprobar en la imagen superior, el servidor obtendrá los puntos GPS del dron que realiza dicha misión en particular, y se le transmitirán para que el cliente pueda dibujar los puntos GPS sobre el mapa. En caso contrario, se le informará que ha ocurrido un error a la hora de obtener dichos puntos GPS.

Por último, tenemos la opción de cambiar el tipo de mapa, con el botón "Tipo de Mapa". Al presionar éste, el cliente abrirá una nueva ventana sobre la "Activity" similar a la que se muestra cuando pulsamos sobre el botón "Misión". Al utilizarlo aparecerá una ventana en la que dispondremos de tres botones mediante los cuales podremos cambiar la vista del mapa entre: satélite, mapa o relieve.

En cuanto a la implementación realizada en la aplicación Android es la siguiente:

```
private void getMapChargeView()
{
    try
    {
        RelativeLayout relativeLayout = (RelativeLayout) getLayoutInflater()
                .inflate(R.layout.map_type_layout,  root: null);
        Button mapView = relativeLayout.findViewById(R.id.mapButton);
        Button satelliteView = relativeLayout.findViewById(R.id.satelliteButton);
        Button reliefView = relativeLayout.findViewById(R.id.reliefButton);

        mapView.setOnClickListener(this);
        satelliteView.setOnClickListener(this);
        reliefView.setOnClickListener(this);

        AlertDialog.Builder alertDialog = new AlertDialog.Builder( context: this);
        alertDialog.setTitle("");
        alertDialog.setView(relativeLayout);
        alertDialog.create().show();
```

Ilustración 102. Mission Activity 22

Tal y como se puede ver la implementación es muy similar a la realizada en el método "openPopupMission", que se ha explicado previamente.

Cabe destacar, que debido a que este paso no necesita ningún proceso en el servidor, al ser independiente del resto de procesos que ocurren en la aplicación, no es necesario realizar ninguna implementación en el servidor.

Por último, se procede a mostrar una serie de diagramas de flujo al igual que se ha venido haciendo a lo largo de este apartado para que se pueda entender el funcionamiento de esta "Activity" de una manera mucho más sencilla y esquemática.

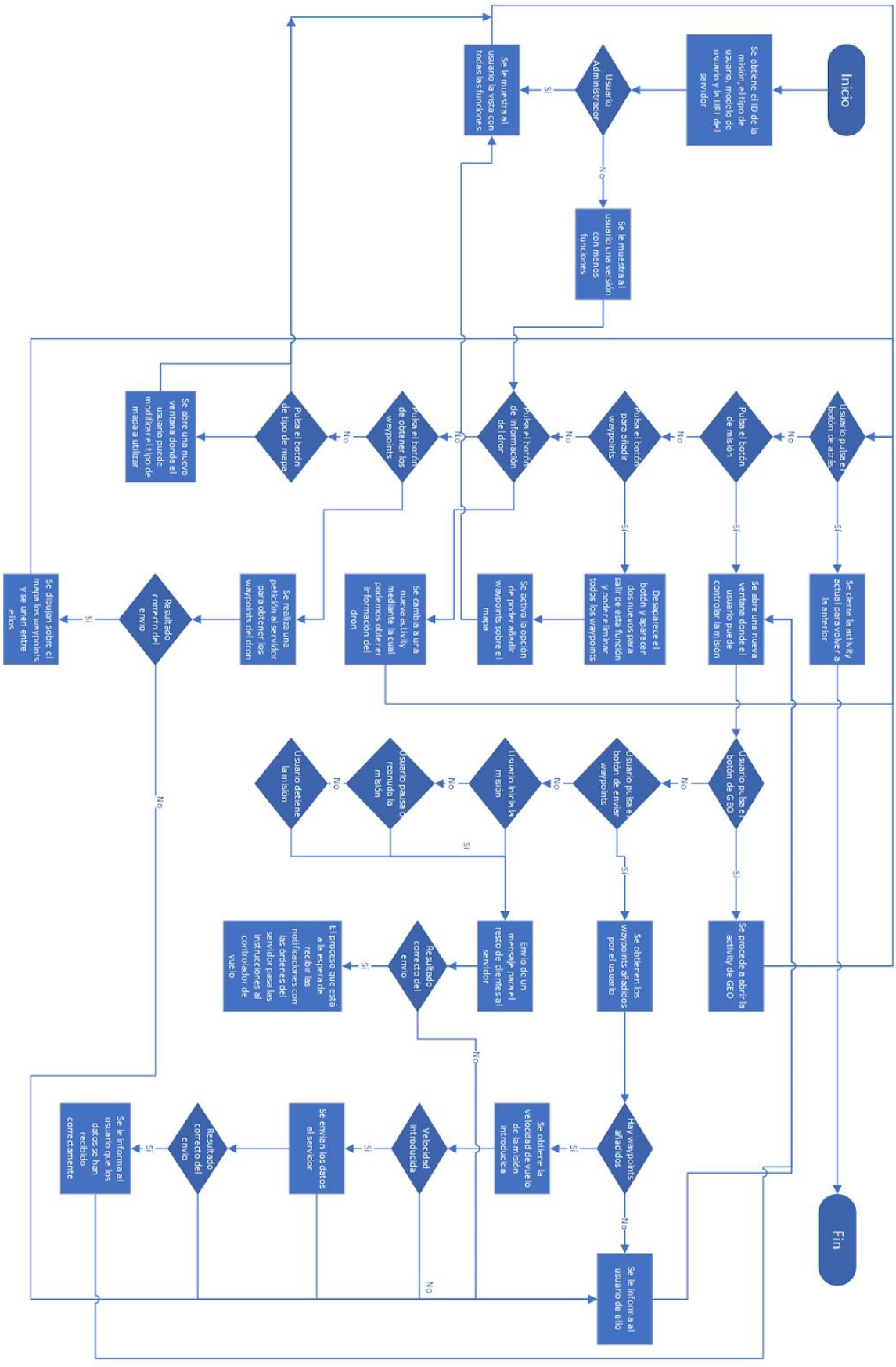


Ilustración 103. Diagrama de flujo Mission Activity

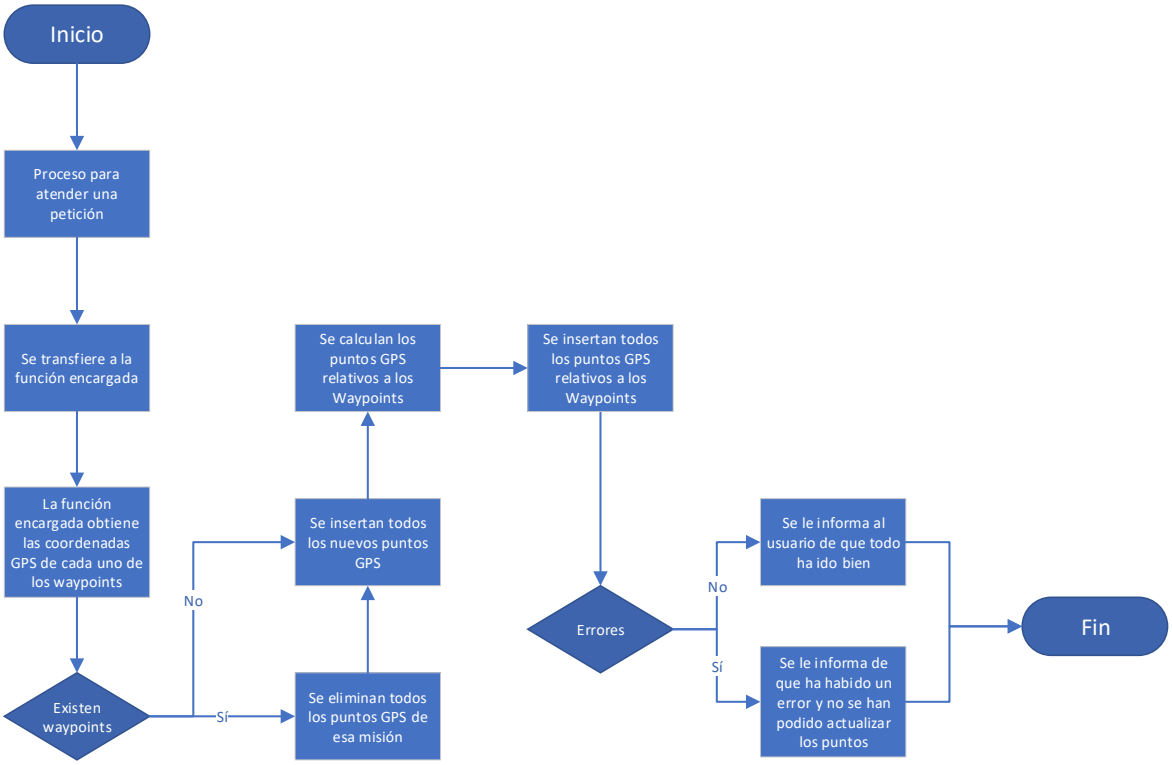


Ilustración 104. Diagrama de flujo Subir Waypoints Servidor 1

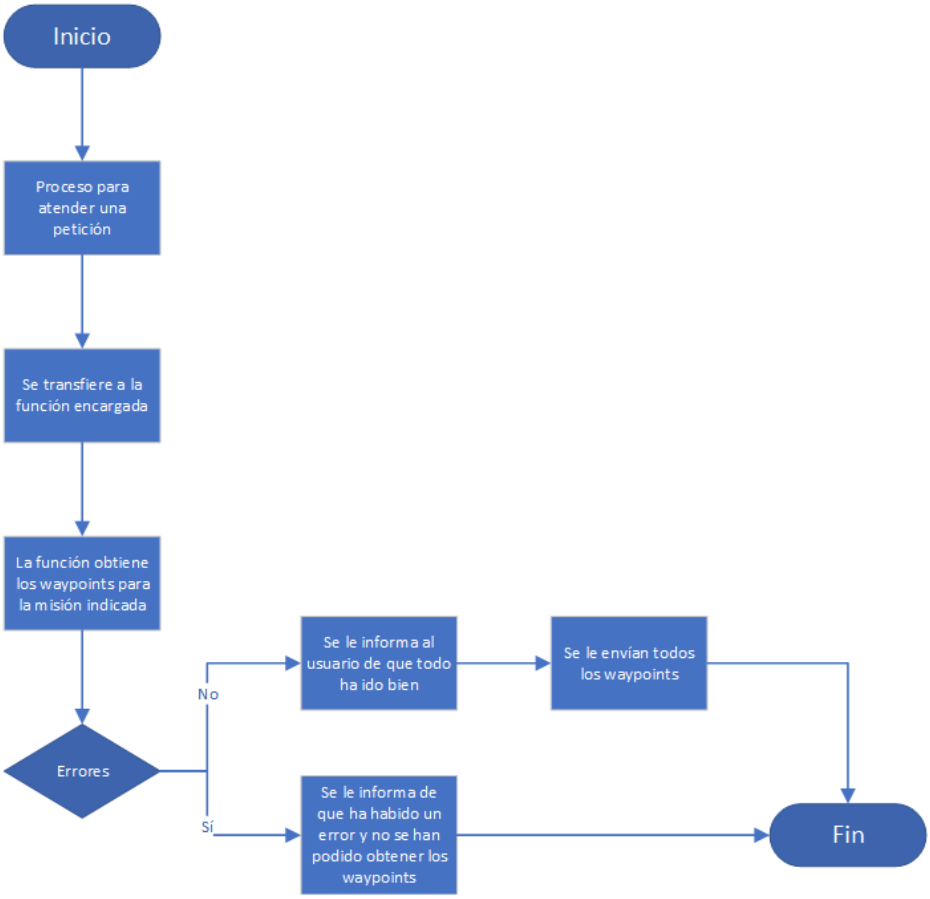


Ilustración 105. Diagrama de flujo Subir Waypoints Servidor 2

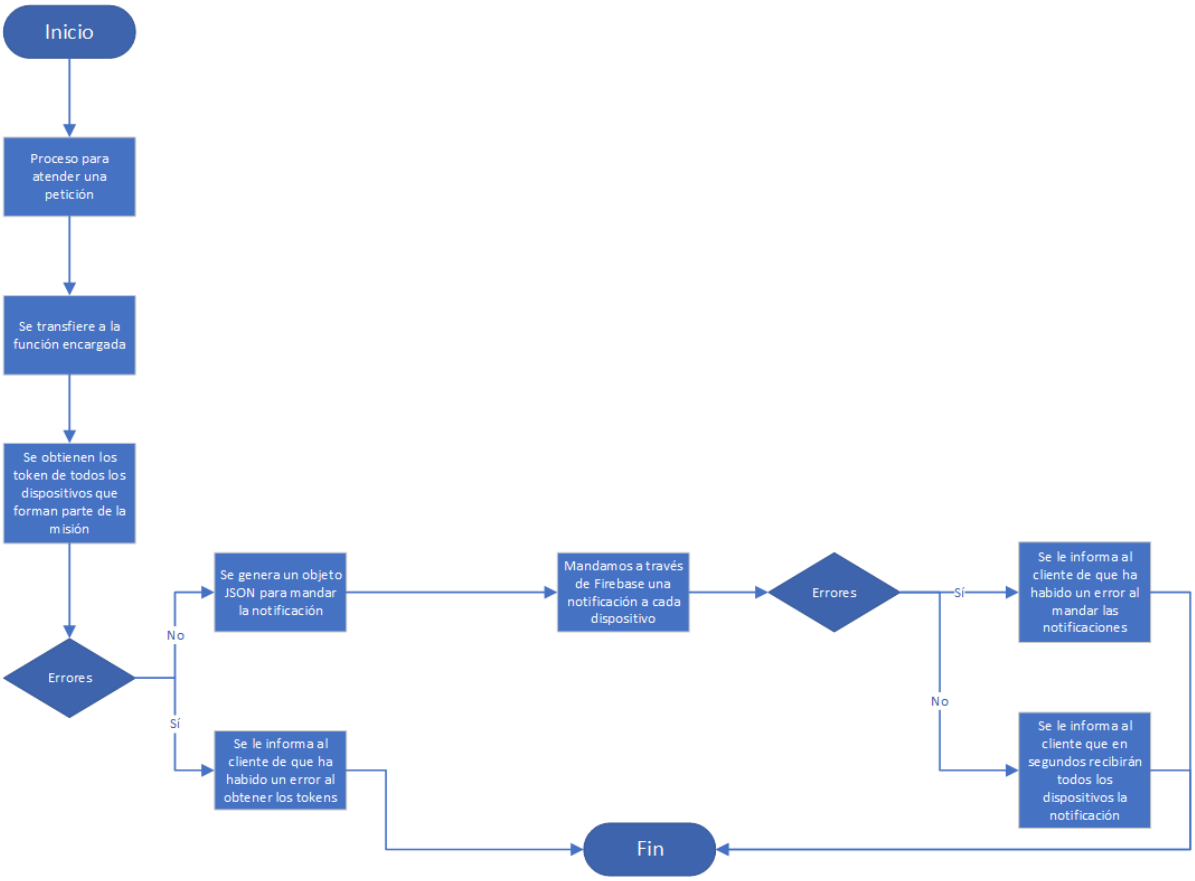


Ilustración 106. Diagrama de flujo Iniciar Misión Servidor

2.6.2. Aclaración de las funcionalidades principales del servidor

A lo largo de este punto analizaremos la estructura de base de datos utilizada para almacenar todo lo relacionado con las misiones y, así, obtener la persistencia necesaria para poder disponer de los datos de una manera sencilla y rápida por parte del servidor. De esta manera, se podrá atender a diversos clientes y con diferentes misiones sin ningún problema. También analizaremos el proceso que se lleva a cabo en el servidor cada vez que éste recibe una nueva petición HTTP. Por último, también analizaremos el proceso que se realiza en el servidor cuando éste se inicia para comprobar que todos los ficheros necesarios se encuentran disponibles y así atender, correctamente, a todas y cada una de las peticiones HTTP que le llegan.

En cuanto al proceso de inicio del servidor tenemos el siguiente:

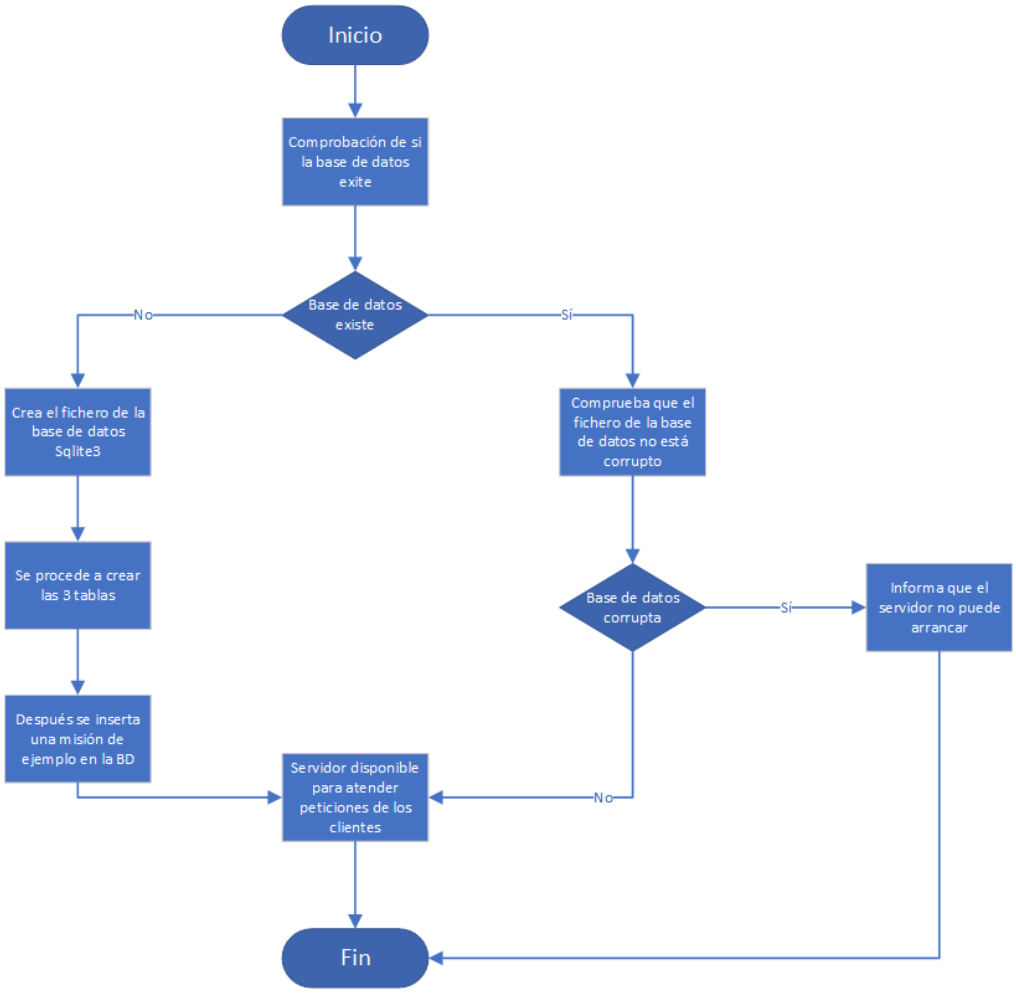


Ilustración 107. Diagrama de Flujo Comprobaciones Iniciales del Servidor

Una vez ejecutamos el comando 'node app.js' el servidor se iniciará y comprobará que el fichero de la base de datos existe. Ese fichero '.sqlite' utilizado por el servidor tiene el nombre de "DB-TFG.sqlite".

Para el caso, de que el fichero indicado exista en la misma ruta en la que se encuentra el fichero 'app.js', el servidor procederá a comprobar si éste se encuentra dañado o corrupto y, si esto ocurre no arrancará, e informará al Administrador mediante un mensaje por consola que existe un problema con la base de datos.

Si por el contrario, la base de datos no se encuentra dañada y, por lo tanto, ésta se puede leer sin problemas el proceso del servidor se lanzará y permanecerá a la espera de recibir peticiones.

En cambio, si la base de datos no se encuentra, bien porque se haya eliminado, o bien porque sea la primera vez que el servidor se ejecuta en el equipo, éste creará un nuevo fichero '.sqlite' en el cual

generará todas y cada una de las tablas necesarias para funcionar. A continuación, se pondrá a la espera de recibir nuevas peticiones API Rest.

En cuanto a la estructura de la base de datos tenemos la siguiente:

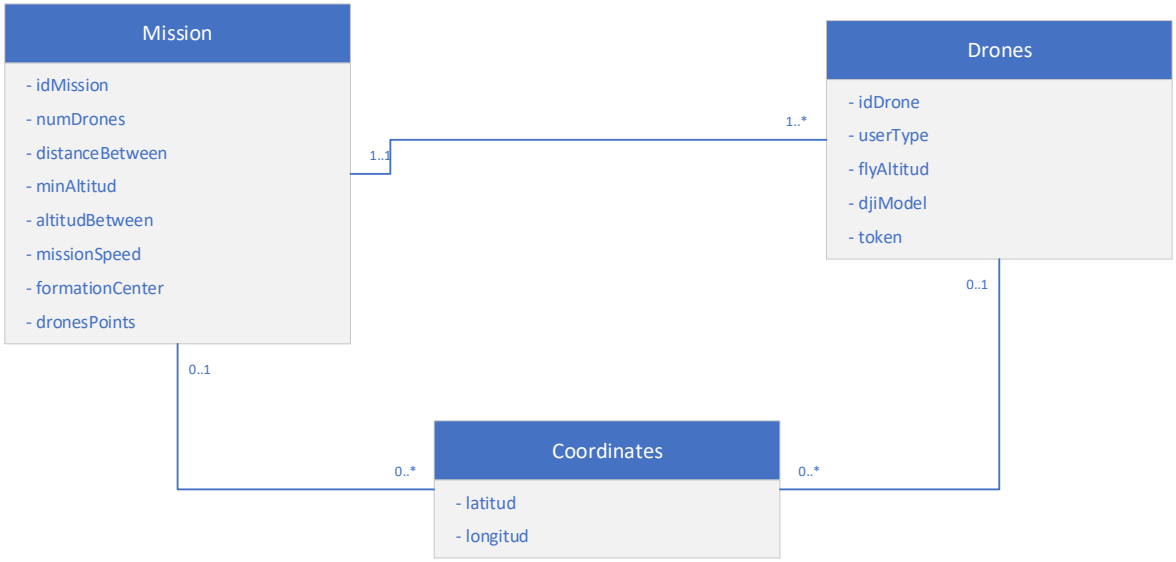


Ilustración 108. Diseño de Base de Datos del Servidor

Gracias a la estructura de base de datos indicada en la imagen superior, podremos almacenar todo lo necesario para realizar una misión, ya que en todo momento el servidor será conocedor de qué "Waypoints" pertenecen a una misión, qué "Waypoints" tendrá que realizar un dron en particular o el número de drones registrados para una misión.

Por último, en cuanto al proceso realizado por el servidor para cada una de las peticiones que a éste le llegan tenemos el siguiente:

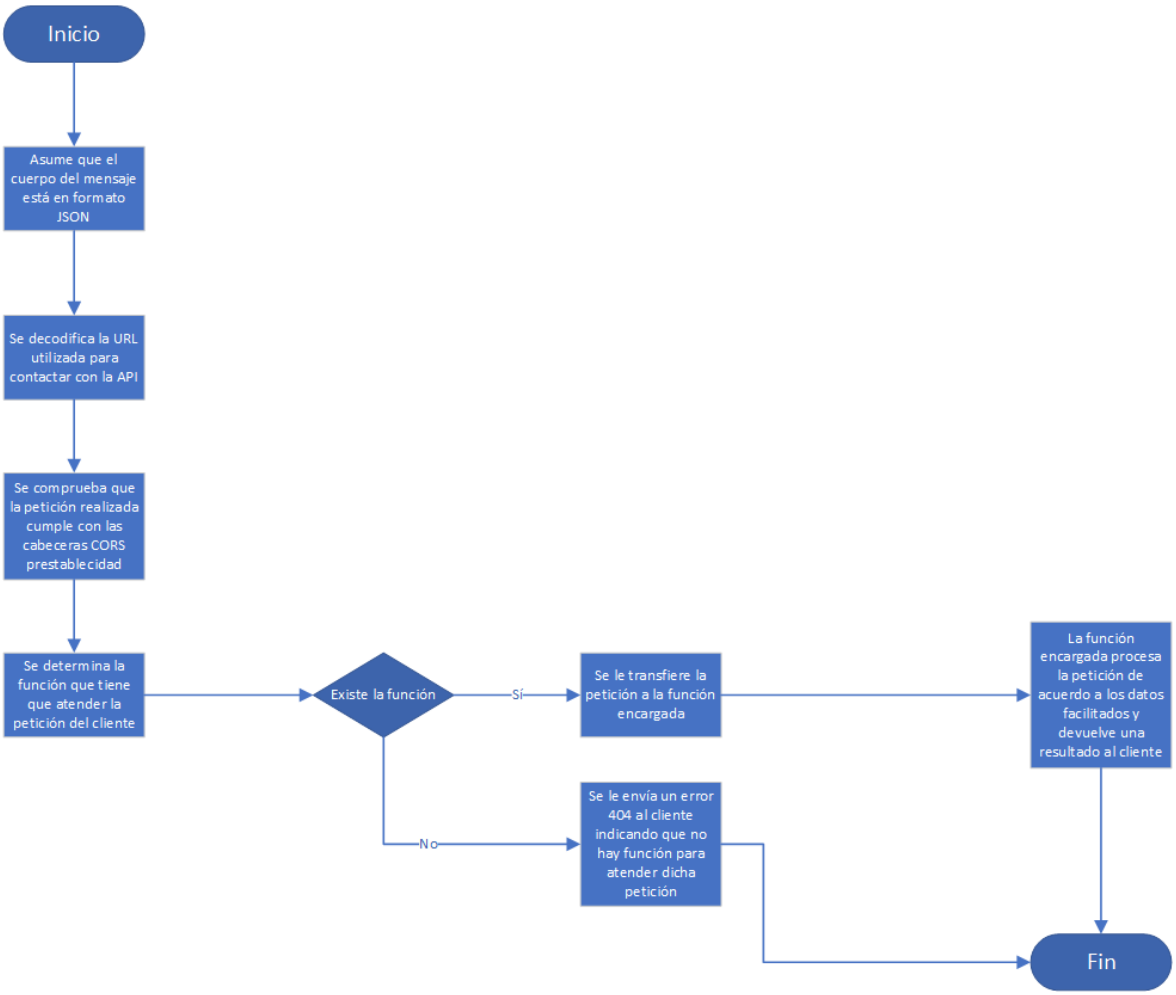


Ilustración 109. Diagrama de Flujo para atender una petición

Cuando una nueva petición llega al servidor las funciones “middleware” de este comprobarán, que el cuerpo del mensaje tiene una estructura JSON, que la URL utilizada para contactar con una función en particular se puede decodificar, y que la petición HTTP cumple con los tipos de cabeceras CORS definidas en la configuración.

Una vez la petición ha sido comprobada y validada por cada una de las funciones “middleware” del servidor se pasará la petición a la función destinada para ello. Cuando la función encargada de atender la petición la procese, le devolverá un mensaje y un código de respuesta HTTP al cliente que la haya realizado, indicando el resultado de la operación una vez procesada su petición.

Por último, destacar que, cuando alguna petición no puede ser atendida por ninguna función, debido a que no existe la encargada de procesar la URL indicada por el cliente y/o la petición HTTP no es del tipo indicado, el servidor devolverá al cliente un error 404.

3. Conclusiones

En primer lugar, procederemos al análisis de los resultados obtenidos una vez el proyecto se ha finalizado, explicando detalladamente cada uno de los principales objetivos. Además, también se procederá a explicar cómo se ha solventado la problemática que existía en cada uno de los desarrollos.

3.1. Análisis de los resultados obtenidos

El principal objetivo es crear una aplicación para dispositivos móviles/tabletas, que permita a cualquier usuario, con independencia, de los conocimientos informáticos que tenga, poder controlar de una manera sencilla e intuitiva un enjambre de drones de la marca DJI, ya que no existen aplicaciones en el mercado, que permitan a los usuarios de drones alcanzar ese nivel de automatización.

Las aplicaciones existentes se limitan a funcionar con un sólo dron automatizado, reduciendo así, el rango de posibilidades, teniendo, además, un coste económico importante.

La aplicación desarrollada ofrece características similares a aplicaciones como Litchi, en la que se pueden realizar misiones en ruta utilizando Waypoints para llevar a cabo acciones sobre ellos.

El desarrollo efectuado permite alcanzar esos Waypoints, utilizando múltiples drones a modo de enjambre, que maximizan las posibles funciones a realizar durante la misión. Un dron puede ser lanzado a una misión con una función específica, y con unas limitaciones de autonomía de vuelo, mientras que la aplicación desarrollada consigue que un grupo organizado de drones pueda realizar funciones repetidas para todos ellos o complementarias, venciendo además la desventaja de la autonomía de vuelo limitada que tiene un solo dron. Y ello, redunda en beneficio de la misión o propósito proporcionando al usuario una mayor eficacia en su cometido.

Hasta tal punto es así, que la aplicación podría incorporarse como una herramienta muy valiosa dentro del tejido productivo. Ello permitiría reducir costes de producción, tiempo de la actividad y evitaría los riesgos del factor humano.

Para que el enjambre de drones sea funcional, es necesario adoptar una serie de medidas: Igual velocidad de vuelo para todos los vehículos, distintas alturas de vuelo y distancias de seguridad entre ellos, para evitar colisiones.

Además de las funcionalidades que se han venido refiriendo en esta memoria, el enjambre de drones abre un sin fin de posibilidades, con una difícil delimitación en estos momentos, ya que tiene tantas y tan variadas aplicaciones que puede servir para usos en actividades presentes y futuras, abriendo campos de investigación y de aplicación para posibles tareas a realizar por estos vehículos.

3.2. Problemas encontrados durante el desarrollo

El tema que se aborda en este trabajo es tan específico y novedoso que no disponíamos de herramientas básicas para el desarrollo cómo pueden ser, “Stack Overflow” o “Github”, ya que tal como hemos dicho, nos adentrábamos en un ámbito muy específico y escasamente desarrollado.

Por ello, ha sido esencial el estudio de la documentación del DJI Mobile SDK, se han seguido las guías oficiales de la marca, que han servido de base para comprender de una forma práctica su aplicación, y así desarrollar el cliente.

Se ha tenido que utilizar servicios relativamente modernos y sin un uso generalizado por parte de los desarrolladores como Firebase, fundamental para la comunicación entre distintos terminales Android, dispositivos que se utilizan para transmitir la orden de inicio de la misión para cada dron de la formación. Además, esta integración ha sido más compleja al tener que realizarla tanto en el cliente como en el servidor.

Debido a la arquitectura de la aplicación, que es cliente-servidor han surgido problemas en la comunicación entre los distintos equipos durante el desarrollo, así como problemas para transmitir la información en formatos que pudieran ser fácilmente procesados por cada uno de los extremos.

La misión a realizar por un dron necesita ser configurada de manera que se conozca parámetros como las coordenadas GPS a visitar, la velocidad y altura de vuelo. Esos parámetros son imprescindibles para el comienzo de la misión, por ello el SDK nos permite de una forma sencilla y eficaz cargar los parámetros en el sistema del vehículo para tenerlos disponibles durante su ejecución.

En el proceso de carga de esos parámetros se ejecutan diversos métodos en distintos hilos, provocando una necesaria comprobación antes de iniciar la misión ya que, de otra forma, en la mayoría de las ocasiones, ésta fallará.

La necesidad de generar diferentes peticiones de forma asíncrona para enviar y recibir datos del servidor ha supuesto un problema al tener que controlar la correcta sincronicidad entre distintos tipos de procesos, es decir, aquellos procesos que envían y reciben contenido del servidor y aquellos propios de la aplicación. Siendo necesario, en muchos casos, la comprobación previa del envío y recepción de datos al servidor, antes de realizar otras tareas.

Debido a las limitaciones encontradas a la hora de utilizar objetos gráficos definidos en el Android SDK ha sido necesaria la creación de objetos personalizados que permitiera el uso de la aplicación por parte de los usuarios de una manera intuitiva.

3.3. Posibles líneas futuras

Uno de los factores más influyentes para el uso regular de una aplicación móvil es que ésta sea sencilla de utilizar por cualquier usuario, desde el mismo momento en el que se instala.

Por ello, algunas de las líneas futuras que se podrían realizar en la aplicación cliente, para permitir una utilización aún más intuitiva por parte de los usuarios, podrían ser las siguientes:

1. Descarga automática de Waypoints que componen una misión en cada uno de los drones, es decir, que cuando el usuario Administrador presione el botón para iniciar la misión, se enviará un mensaje a todos los clientes, para que procedan a solicitar al servidor los Waypoints asociados al dron conectado. Una vez los Waypoints de cada uno de los drones se encuentren descargados, se debería iniciar la misión de manera automática.

2. En lugar de utilizar un desplegable, con únicamente tres posibles velocidades para realizar la misión, se debería sustituir dicho desplegable por otro tipo de objeto gráfico, que permitiera de una manera intuitiva especificar la velocidad de vuelo de la misión.

3. Modificación de la forma de introducir aspectos como: altura mínima de vuelo o distancia entre drones, debido a que podría resultar confuso para algunos usuarios en una primera experiencia con la aplicación.

4. Durante el desarrollo de la misión podría resultar útil para los usuarios que se pudiera visualizar, sobre el mapa de Google Maps aspectos relevantes sobre los drones, tales como: la velocidad y altura actual o el porcentaje restante de batería en el dron.

5. Incluir la posibilidad de seleccionar una sección de terreno para que los drones, utilizando una formación puedan mapear dicha sección desde distintas alturas y perspectivas.

6. Añadir la posibilidad de seleccionar entre un abanico de acciones a realizar por los drones cuando lleguen a un Waypoints.

7. Mantener al usuario informado en todo momento sobre el tiempo restante estimado para completar la misión.

8. Incluir alguna medida de seguridad adicional como puede ser, la comprobación de los niveles de batería bajos, para evitar que cualquier dron salga de la formación de forma inapropiada en su regreso al punto de despegue.

3.4. Revisión de los objetivos del proyecto

En cuanto a los objetivos definidos al inicio de este proyecto, destacar que, todos han sido alcanzados, y además se han establecido los parámetros para futuras actualizaciones o mejoras. Así:

1. Se han aplicado los conceptos adquiridos durante los distintos cursos del grado en Ingeniería Informática por la Universidad de Alicante, en asignaturas como: Programación 3, Redes de Computadores, Interconexión de Redes, Desarrollo de Aplicaciones para Internet, Diseño de Bases de Datos, Administración de Sistemas Operativos en Redes de Computadores, Sistemas Distribuidos o Análisis y Especificación de Sistemas Software.

2. Durante el proyecto se han adquirido los conocimientos necesarios para saber utilizar herramientas como el Mobile SDK para automatizar múltiples tareas con UAVs, ya sea utilizando un único dron o múltiples de ellos.

3. Se han adquirido los conocimientos necesarios para poder desarrollar aplicaciones desde cero para dispositivos Android, incluyendo la implementación de la parte lógica, como de vistas gráficas utilizables para distintos dispositivos de distinto tamaño y resolución.

4. Se ha comprendido el uso y aplicación de SDKs como el de Google Maps para Android, con la intención de integrar dentro de aplicaciones propias los mapas de Google Maps.

5. Se han conocido y aplicado nuevos servicios modernos como Firebase de Google, para poder notificar a usuarios de una aplicación desarrollada por uno mismo de una manera sencilla, mediante el uso de sus propios servidores.

6. Se han utilizado los conocimientos adquiridos en el grado para crear una aplicación con la arquitectura cliente-servidor.

7. Se han conocido nuevas herramientas muy útiles como el simulador de DJI, para poder comprobar el correcto funcionamiento de aplicaciones para drones de la marca. Para ello, es necesario el uso y conexión a drones reales.

8. Se ha investigado y analizado el procedimiento o los posibles procedimientos para crear una aplicación, que permite controlar un enjambre de drones desde un único terminal Android, utilizando los conocimientos adquiridos en asignaturas como Redes de Computadores.

9. Por último, el servidor se ha desarrollado utilizando los frameworks "express" y "knex" en un servidor en Node JS utilizados durante el grado, para atender todas las peticiones HTTP de cada uno de los distintos clientes Android.

La aplicación para el uso de enjambres puede ser la base para el desarrollo de futuros proyectos que puedan implementar o facilitar la realización de un sin fin de actividades. Es una aplicación abierta a que futuros desarrolladores la puedan aplicar a sus investigaciones en distintos campos.

3.5. Demostración funcionamiento de la aplicación

Pasos realizados en la simulación
1. Instalación de la aplicación en dispositivo Android. 2. Ejecutar la aplicación y proporcionar permisos de usuario. 3. Espera hasta registrar la aplicación en los servidores de DJI. 4. Encendido del UAV y del control remoto. 5. Enlazar el control remoto y el terminal Android. 6. Conectar por USB el vehículo al PC. 7. Introducir las coordenadas iniciales en el simulador. 8. Inicio del servidor. 9. Habilitar GPS del móvil. 10. Creación de una nueva misión. 11. Registro de un usuario Administrador en la nueva misión. 12. Modificación de los valores añadidos por defecto a la misión creada. 13. Establecimiento de los Waypoints. 14. Envío de los Waypoints al servidor. 15. Descarga de puntos GPS relativos a los Waypoints. 16. Inicio de la misión.

Durante el desarrollo del presente apartado se procederá a explicar el procedimiento de configuración de la aplicación para que pueda llevar a cabo una misión de forma correcta. Debido a las limitaciones por el estado de alarma, a causa de la pandemia provocada por el coronavirus (Covid-19), se procederá a realizar esta simulación en un entorno cerrado.

Ésta se realizará utilizando el simulador de DJI, ya mencionado en apartados anteriores de la presente memoria. Además, debido a las limitaciones en el número de drones disponibles para el experimento, ya que contamos sólo con uno, se procederá a explicar la posición a la que se dirigirá cada dron, dependiendo de su ID y de la misión.

Por lo tanto, la simulación se deberá realizar de manera escalonada, es decir, se simulará el comportamiento de cada componente de la formación por separado, al no disponer de varios drones para poder realizarla en paralelo.

Una vez aclarado lo anterior, debemos instalar la aplicación con el APK generado por Android Studio. Cuando dispongamos de la aplicación ya en nuestro dispositivo Android, procederemos a ejecutarla y nos aparecerá lo siguiente:

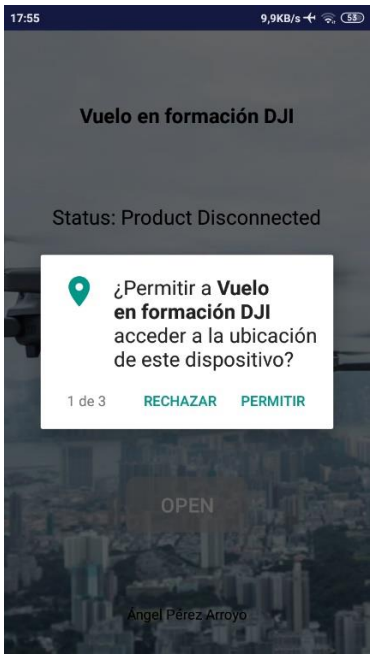


Ilustración 112. Permiso de ubicación

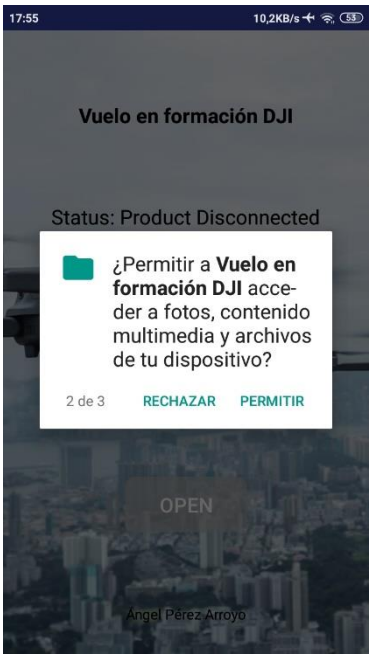


Ilustración 110. Permiso contenido multimedia

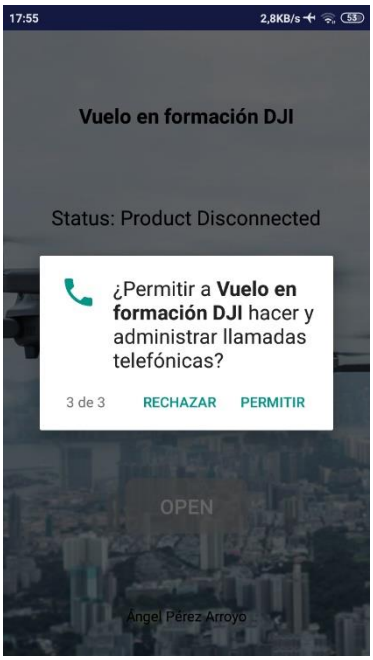


Ilustración 111. Permiso de llamadas

Cabe destacar que los permisos anteriores tan sólo se solicitarán en la primera ocasión, que se ejecute la aplicación, ya que el sistema operativo los almacenará, lo que permitirá disponer de dichos permisos en repetidas ocasiones sin solicitarlos continuamente al usuario.

El primer permiso es necesario para poder geolocalizar la posición del usuario del terminal, con la finalidad de que le aparezca su posición, en el mapa de Google Maps, en el centro de la pantalla. El segundo permiso es solicitado con la finalidad de poder almacenar las fotos y/o vídeos que se realicen con el dron. Por último, el tercer permiso es requerido por el Mobile SDK para funcionar correctamente.

Si el usuario no aprueba alguno de los 3 permisos anteriores, la aplicación no iniciará ningún proceso ya que, de acuerdo con la política utilizada durante el desarrollo, la aplicación tan sólo funcionará si todos los permisos se encuentran aprobados. En caso contrario, se le volverán a solicitar, al usuario los que resten hasta que apruebe todos.

Cuando se dispongan de todos los permisos aprobados, se mostrará la vista “Connection Activity”, tal y como se muestra en la imagen inferior:

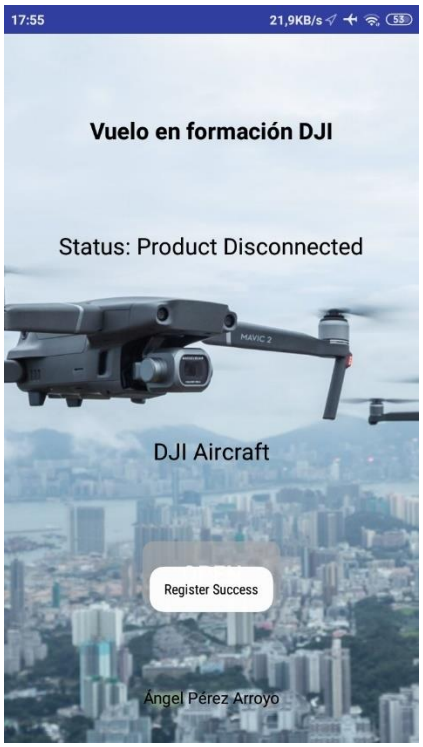


Ilustración 113. Connection Activity Simulation

A continuación, la aplicación intentará registrarse en los servidores de DJI. Para que éste y otros muchos procesos de la aplicación funcionen correctamente, debemos disponer de conexión a Internet, ya sea vía Wifi o vía datos móviles. En caso contrario, muchos procedimientos necesarios para ejecutar la aplicación fallarán.

Una vez registrada la aplicación, el siguiente paso será conectar el dron que deseemos emplear y su correspondiente control remoto. A continuación, conectaremos mediante USB el terminal Android al control remoto de la aeronave.

Después, esperaremos hasta que se establezca una conexión entre el control remoto y el UAV. Cuando dicha conexión se haya llevado a cabo, el botón "Open" se encontrará disponible.

Ilustración 114. Connection Activity Disponible Simulation

Después podremos pulsarlo para pasar a la siguiente "Activity", ya que la conexión estaría establecida entre el terminal Android, el control remoto y, a su vez, éste con el dron.

Antes de continuar es recomendable conectar el dron al ordenador a través de un puerto USB para poder utilizar el simulador y, después abriremos la aplicación “DJI Assistant 2” para poder acceder al simulador.

Entonces veremos una ventana como la que se muestra a continuación:

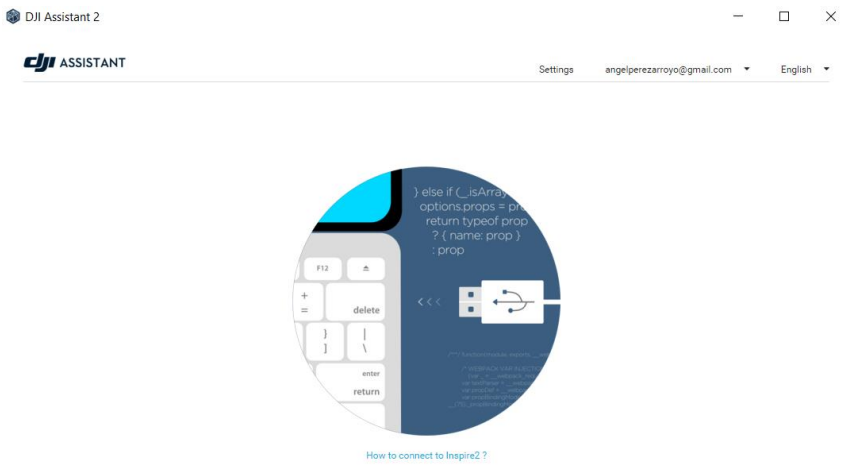


Ilustración 115. DJI Assistant

Debemos esperar unos segundos hasta que el dron también sea detectado por el programa. Cuando esto ocurra, la ventana cambiará y se mostrarán los drones detectados por la aplicación. En este caso el modelo a utilizar va a ser un “Mavic Air”.

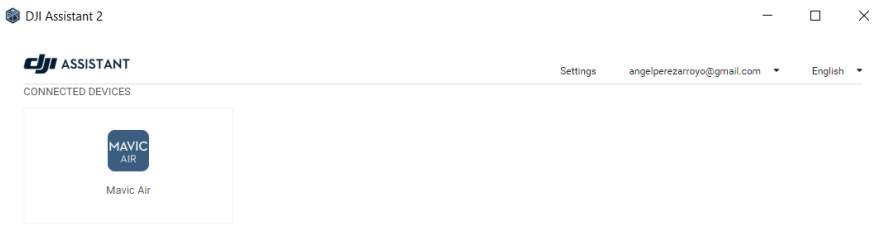


Ilustración 116. DJI Assistant Mavic detectado

El siguiente paso será simplemente hacer ‘click’ sobre la opción del dron, que deseemos utilizar.

Cuando hayamos realizado el procedimiento anterior, nos aparecerá una ventana con algunas opciones en un panel lateral, que pueden variar dependiendo del modelo. Debemos localizar la opción del simulador y acceder a ella.

Ilustración 117. DJI Assistant Simulador

Finalmente haremos 'click' sobre el botón "Open" dentro de la opción del simulador para acceder a la siguiente ventana, donde introduciremos las coordenadas iniciales que deseemos. En mi caso, he utilizado las de mi ubicación, aunque por supuesto, se podrían utilizar cualquier otra ubicación sin problema alguno.

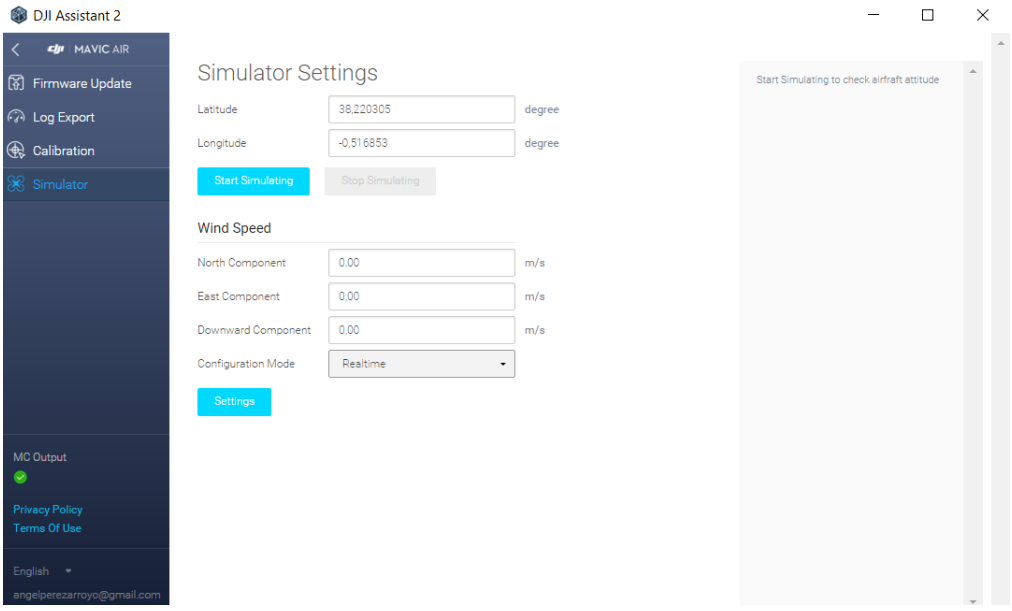


Ilustración 118. DJI Assistant Coordenadas

No he iniciado la simulación hasta llegar a la "Mission Activity", pero no existiría inconveniente en hacerlo antes. El motivo por el que, he tomado esta decisión es por la temperatura que alcanza el dron cuando realiza simulaciones y por el consumo de la batería.

El siguiente paso, que debemos realizar es el inicio del programa, que se ejecuta en un servidor para determinar los aspectos de la misión para cada uno de los drones utilizados. Para iniciar dicha aplicación debemos emplear el comando, que se muestra en la imagen inferior.

```
angel@angel-VirtualBox:~/Documentos/PROYECTO-TFG$ node app.js
Aplicación lanzada en el puerto 9090!
```

Ilustración 119. Arranque Servidor

Una vez alcanzado este punto sería recomendable tener el GPS del terminal utilizado activado, tal y como se puede ver en la siguiente imagen. Este procedimiento resultará muy útil, ya que al cargar la principal "Activity" de la aplicación, aparecerá la posición del usuario en tiempo real en el centro del mapa de Google Maps.

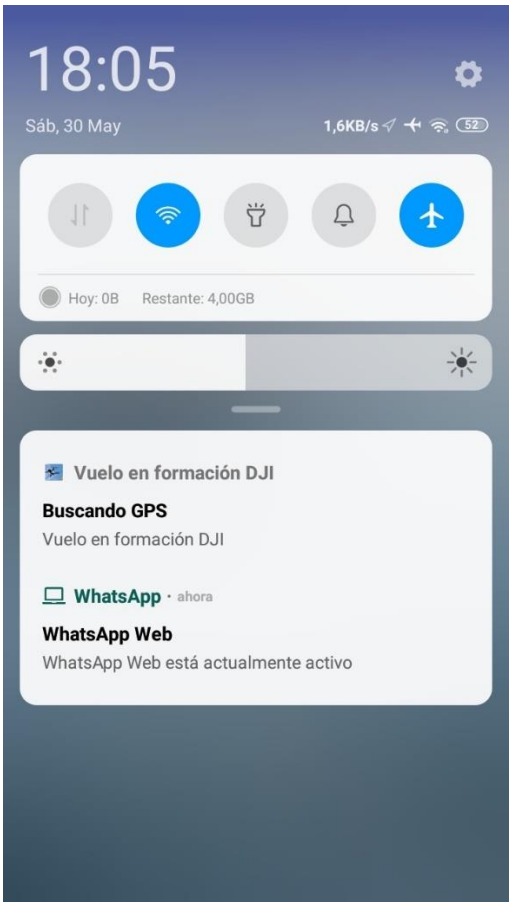


Ilustración 120. Buscando GPS

Cuando hayamos completado los pasos previos, y nos encontremos en la "Create Mission Activity", indicaremos si deseamos crear una nueva misión o acceder a una creada ya previamente

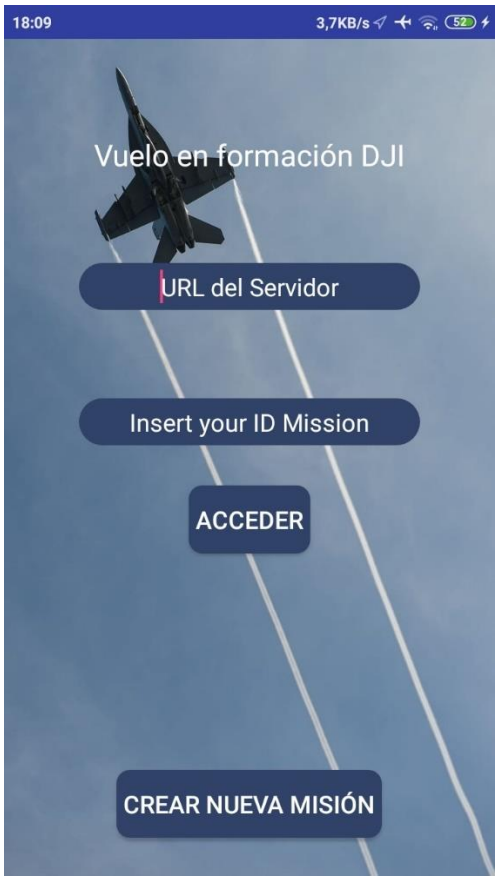


Ilustración 122. Create Mission Primera Vez

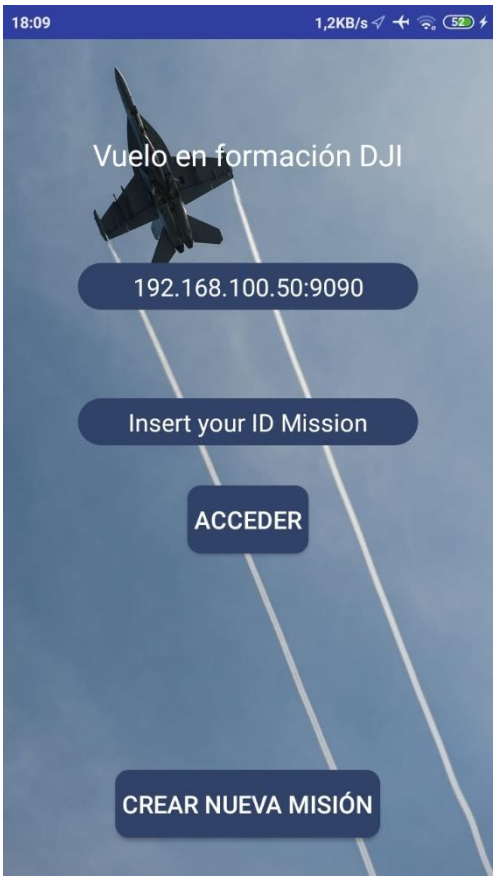


Ilustración 121. Create Mission Después

Además, tal y como se puede apreciar en las imágenes 121 y 122, la primera vez que iniciemos la aplicación debemos introducir la URL del servidor, que hemos inicializado previamente. En el resto de las ocasiones, la aplicación automáticamente escribirá la última URL válida empleada.

En la parte inferior, encontraremos la posibilidad de introducir el ID de la misión a la que queremos acceder o la posibilidad de crear una nueva misión. Para el ejemplo que se está mostrando se ha creado una nueva misión con un primer terminal, y con un segundo se ha accedido a la misión creada previamente.

Posteriormente, en el servidor podremos encontrar una nueva fila en la tabla “mission”. Cabe recordar que la primera misión, que aparece se crea por defecto cuando se genera la base de datos por parte del servidor. Por lo tanto, la misión que se acaba de crear es la que tiene ID 2.

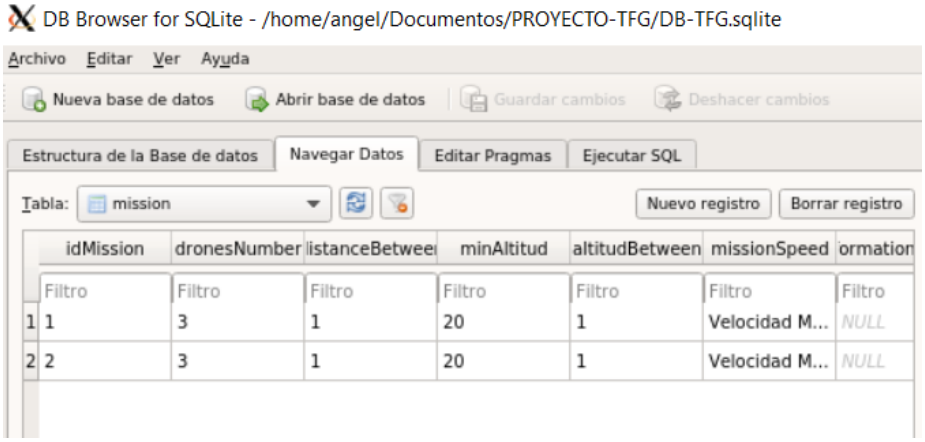


Ilustración 123. Tabla Misión Demostración

Si el proceso de creación de una nueva misión ha sido correcto, la aplicación cliente proporcionará acceso a la siguiente "Activity" donde nos podremos registrar como usuarios o administradores de la misión.

Ilustración 125. Main Activity Demostración

Ilustración 124. Main Activity Administrador Demostración

Dicha comprobación de registro la podremos verificar en la tabla “drones”, en la que veremos que se ha registrado un nuevo usuario Administrador, el cual se encuentra asignado a la misión con ID 2, que es la que acabamos de crear.

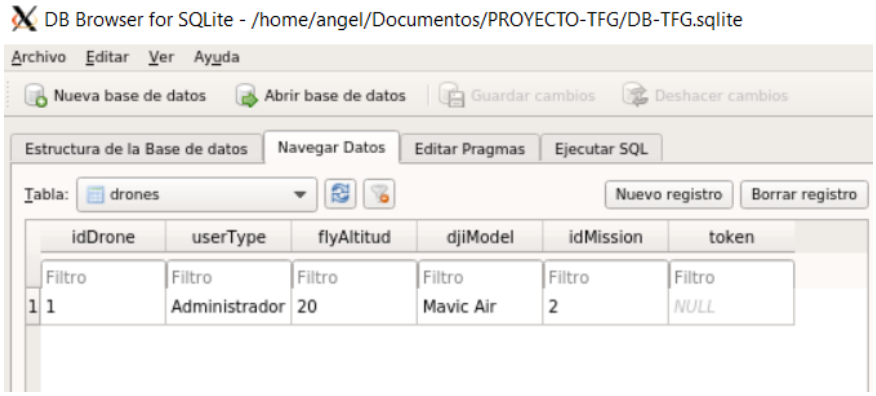


Ilustración 126. Registro dron Demostración

En caso de haberse realizado correctamente el registro en la base de datos del servidor, tal y como se ha mostrado en la imagen superior, se procederá a cargar la “Fly Register Activity”. Debemos recordar, que dicha “Activity” tan sólo será accesible para los usuarios Administradores, ya que nos permite personalizar la misión conforme a nuestras necesidades o gustos.

Ilustración 128. Fly Registry Demostración

Ilustración 127. Fly Registry Rellenada Demostración

Una vez el usuario Administrador de una misión haya rellenado todos los campos solicitados en la "Activity" mostrada en las dos imágenes superiores, éste deberá de dibujar la formación de los drones durante el desarrollo de la misión, tal y como se puede ver en la imagen número 126.

Finalmente, el usuario presionará el botón "Actualizar" y se enviarán todos los datos al servidor para poder determinar la posición de cada uno de los drones de la misión, así como, su altura de vuelo.

Los datos introducidos en el cliente de la aplicación son enviados y almacenados en la base de datos del servidor, tal y como se puede apreciar en la siguiente imagen.

Ilustración 129. Actualizar misión Demostración

Posteriormente, el servidor informará al cliente, que puede proporcionar al usuario el acceso a la vista principal de la aplicación ("Mission Activity").

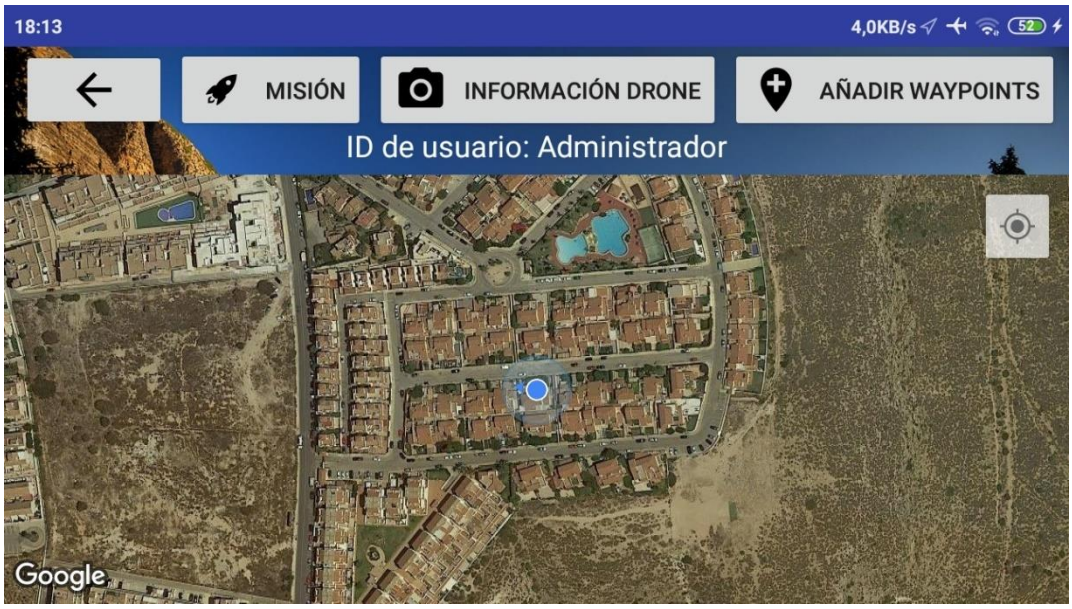


Ilustración 130. Mission Activity Demostración

Una vez se ha cargado la vista y el mapa de la “Mission Activity”, podremos comprobar que la aplicación ha obtenido nuestra posición mediante el uso del GPS del terminal Android empleado, y ha colocado la posición del usuario en el centro del mapa.

Además, podemos optar entre 3 tipos de mapas disponibles en la aplicación, tal y como se puede apreciar en la siguiente imagen, y además estas opciones pueden ser intercambiables.

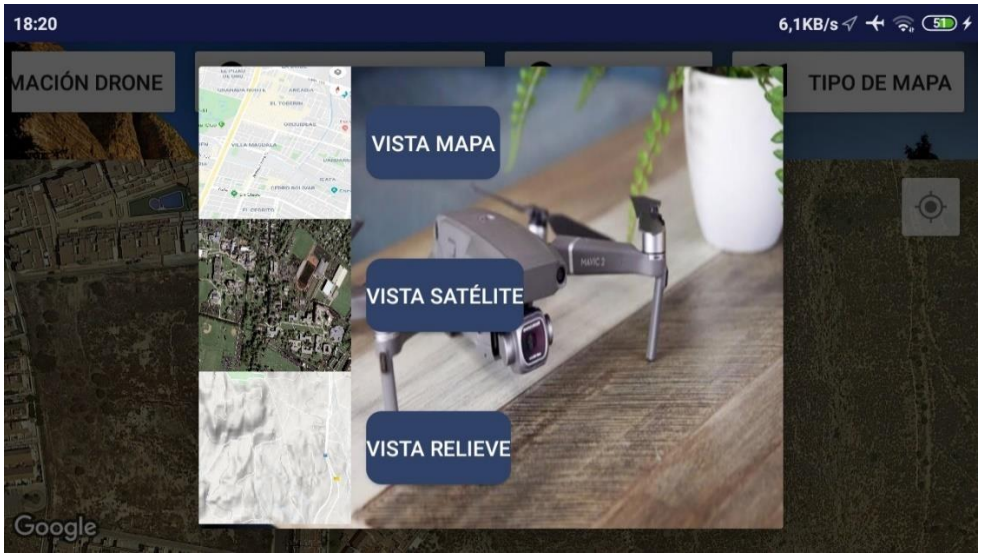


Ilustración 131. Cambio de mapa Demostración

Las siguientes imágenes muestran la perspectiva de relieve y de mapa tradicional.

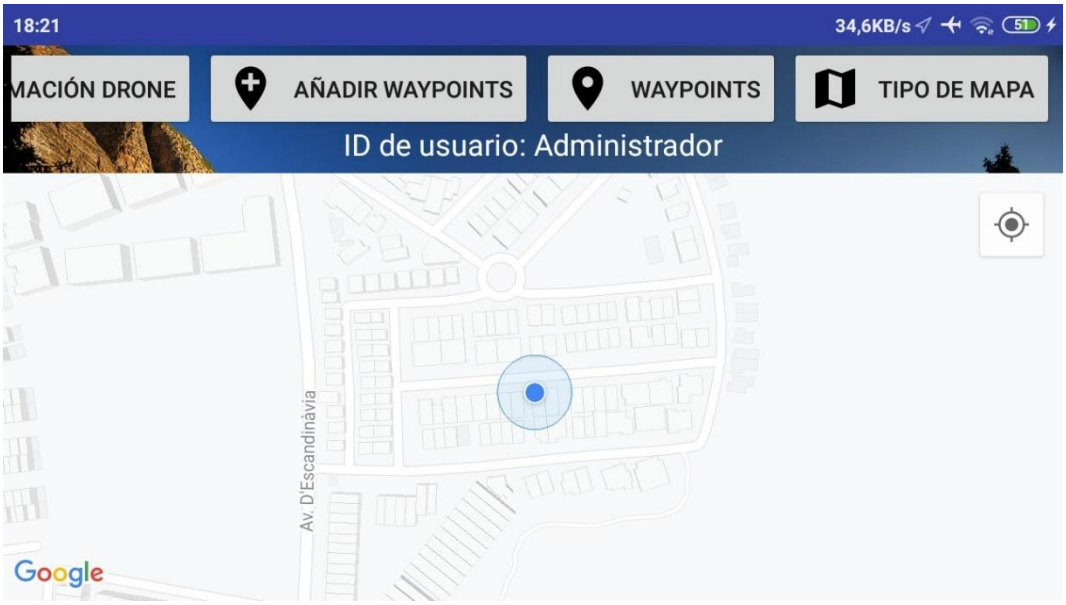


Ilustración 132. Vista mapa Demostración

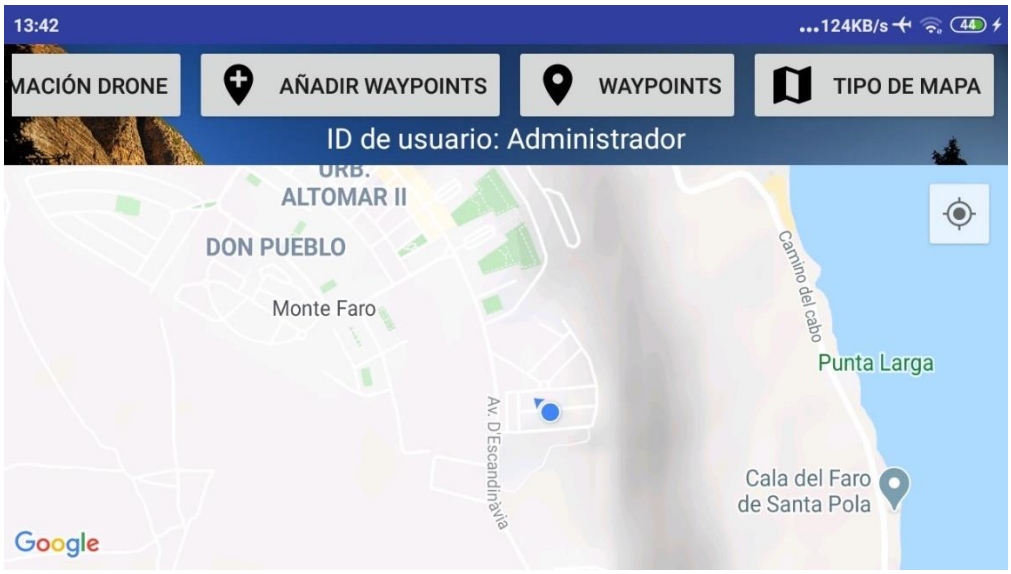


Ilustración 133. Vista relieve Demostración

El estado de los sensores de la aeronave puede ser verificado por el usuario, simplemente, presionando el botón de "Información Dron", y accederá a la vista que se muestra a continuación, donde se pueden verificar distintos valores sobre la aeronave conectada a ese terminal.

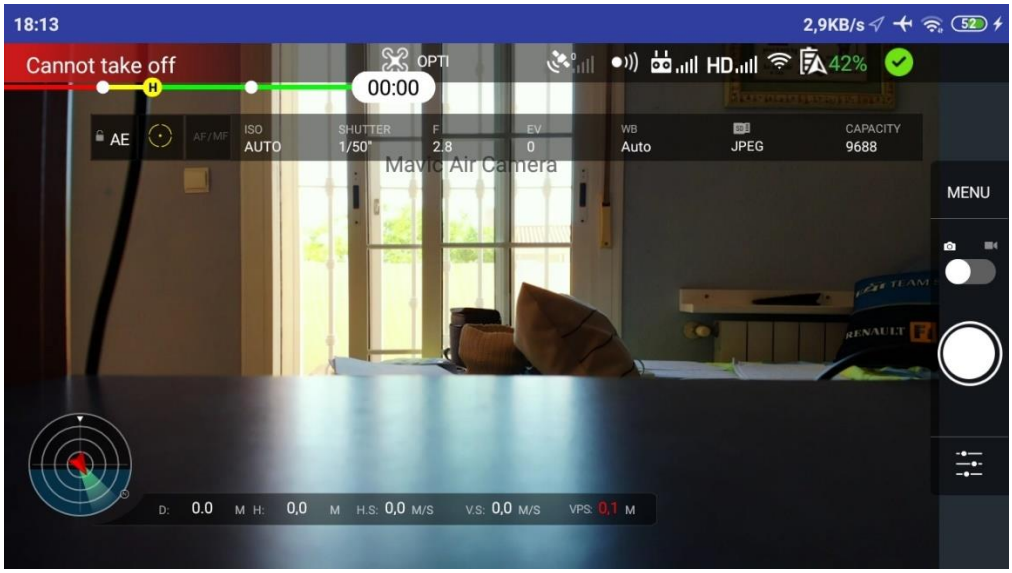


Ilustración 134. Información dron Demostración

A continuación, para definir una nueva misión, habilitaremos la opción añadir Waypoints de tal forma que cada vez que toquemos la pantalla del terminal se añadirá un nuevo marcador. De esta manera, el usuario puede ir añadiendo tantos puntos como desee que se lleven a cabo en la misión, con un mínimo de 1 y un máximo de 99.

Ilustración 135. Añadiendo Waypoints Demostración

Cuando ya se hayan añadido todos los puntos deseados podremos, deshabilitar la función de añadir Waypoints y enviaremos dichos puntos al servidor.

Para enviar los puntos GPS de cada uno de los Waypoints, nos dirigiremos a la opción de misión, donde nos aparecerá la siguiente ventana:

Ilustración 136. Misión Demostración

Una vez allí, debemos introducir la velocidad con la que deseamos ejecutar la misión. Para el caso de esta demostración, se ha seleccionado una velocidad alta, que permitirá a cada uno de los drones alcanzar una velocidad máxima de vuelo de 12m/s.

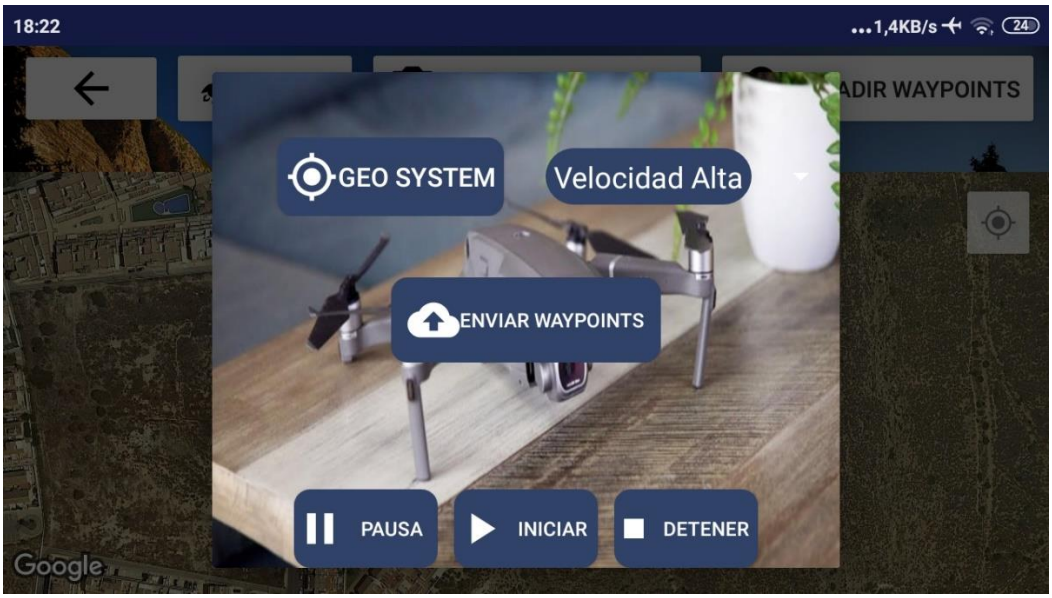


Ilustración 137. Enviar Waypoints Demostración

Finalmente, procederemos a presionar el botón para enviar los Waypoints. En el caso de haber enviado todos los puntos GPS correctamente al servidor, se indicará en la pantalla, tal y como se puede ver en la imagen número 138. Esta información resulta muy útil, ya que nos mostrará que todos los drones registrados para esa misión ya disponen de sus correspondientes puntos GPS.

Ilustración 138. Envio de Waypoints Demostración

Esta información la podremos comprobar si nos dirigimos a la tabla “coordinates” de la base de datos del servidor, donde veremos los resultados obtenidos por el algoritmo, que se encarga de calcular cada uno de los puntos relativos a los Waypoints.

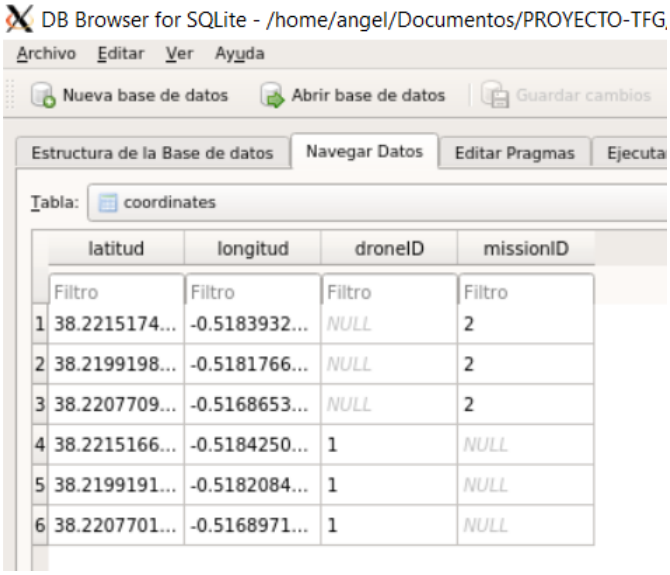


Ilustración 139. Cálculo de coordenadas relativas Demostración

Debido a que la distancia indicada ha sido de tan sólo 3 metros, tal y como se puede apreciar, las coordenadas GPS introducidas por el usuario y las calculadas por el algoritmo, difieren por muy poco. A través de herramientas como Google Maps, se ha podido verificar que entre dos puntos GPS, efectivamente, existe la distancia indicada por el usuario.

El siguiente paso consistirá en descargar los Waypoints correspondientes al dron conectado. Para ello, simplemente será necesario presionar el botón de Waypoints y se iniciará la descarga de manera automática. Cuando esto haya finalizado correctamente, la aplicación procederá a dibujar sobre el mapa de Google Maps, los puntos GPS recibidos por parte del servidor y a unirlos entre ellos.

Además, tal y como se puede comprobar en la imagen inferior, si deseamos conocer el orden en el que cada dron visitará cada uno de los Waypoints, debemos presionar sobre los marcadores y aparecerá una etiqueta en la que veremos el orden en el que se visitará ese Waypoint en concreto.

Ilustración 140. Descarga Waypoints Demostración

Antes de ejecutar la misión iniciaremos el proceso de simulación en el dron que al principio de la explicación hemos conectado al ordenador. Para ello, simplemente haremos 'click' sobre el botón "Start Simulation", que encontraremos dentro del apartado del simulador de la herramienta DJI Assistant.

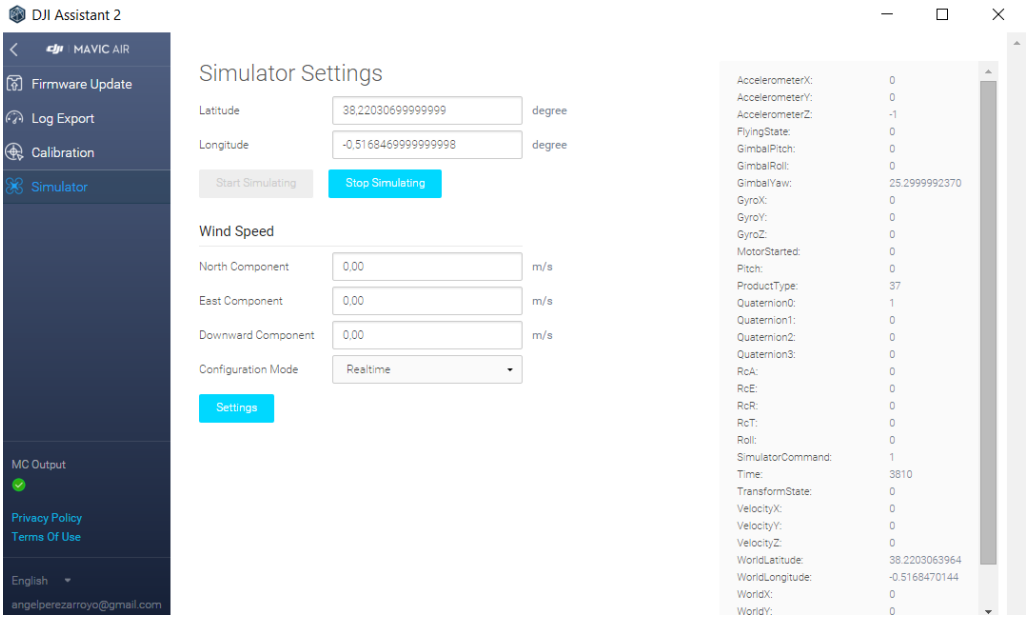


Ilustración 141. Inicio Simulación Demostración

Una vez iniciada la simulación en el dron, debemos observar cómo de manera automática nos aparece un nuevo elemento sobre el mapa. Éste tiene forma de avión y será el icono, que nos irá indicando en todo momento la ubicación del dron sobre el mapa mientras se ejecuta la simulación.

Ilustración 142. Dron sobre mapa Demostración

Cuando dispongamos de nuestros Waypoints y nuestro dron dibujados sobre el mapa integrado dentro de la aplicación, podremos iniciar la misión. Para ello, debemos dirigirnos al botón “Misión” nuevamente, pero en esta ocasión presionaremos el botón de iniciar misión y se enviará un mensaje a través de Firebase que será recibido por todos los terminales asignados a una misión.

En el apartado de notificaciones de nuestro dispositivo aparecerá una notificación indicando que se debe iniciar la misión, la aplicación comienza a transmitir toda la información necesaria al dron, para que éste pueda comenzar a volar tal y como se puede ver en las siguientes imágenes:

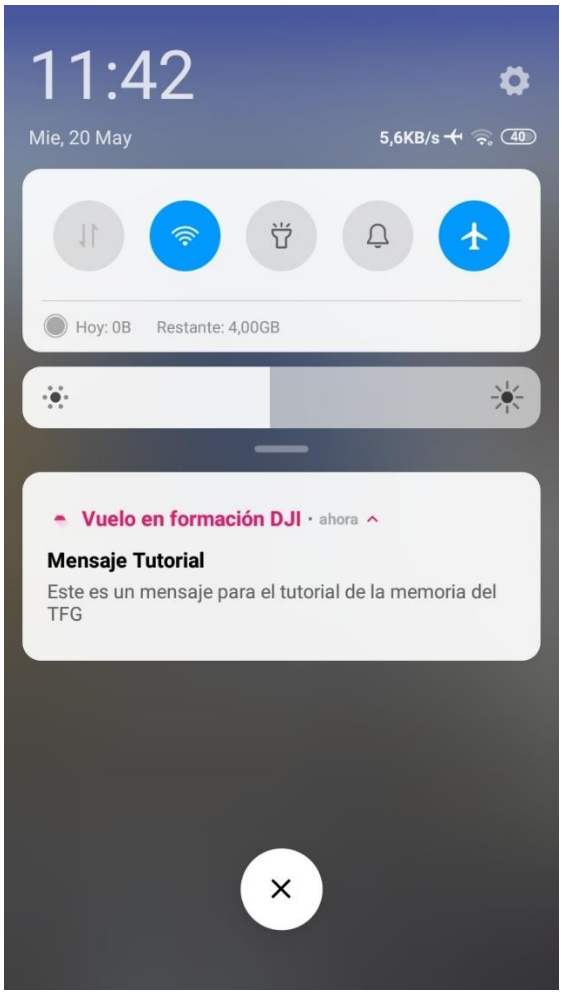


Ilustración 143. Inicio Misión Notificación Demostración

Ilustración 144. Inicio Misión Demostración

Cuando todos los Waypoints, la altura de vuelo, la velocidad máxima de vuelo y las tareas a realizar sobre cada uno de los puntos GPS indicados se hayan cargado en la memoria del dron, veremos como éste en el simulador comienza a alzar el vuelo hasta la altura indicada.

La altura de vuelo para este dron es de 50 metros tal como se puede ver en la imagen inferior.

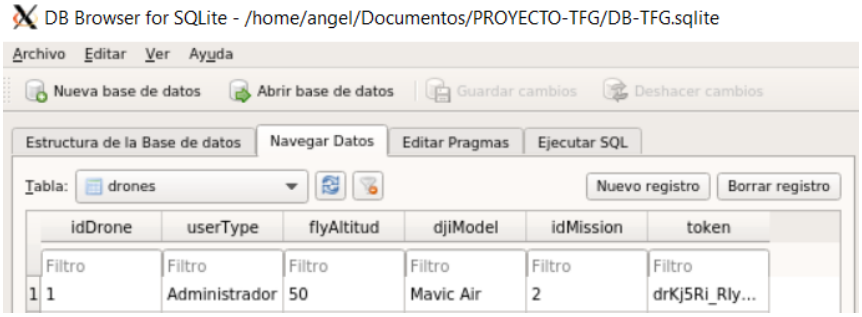


Ilustración 145. Altura vuelo dron Demostración

En las siguientes imágenes se puede comprobar como el dron en primer lugar asciende hasta los 50 metros de altura antes de iniciar su camino hasta el primer Waypoint.

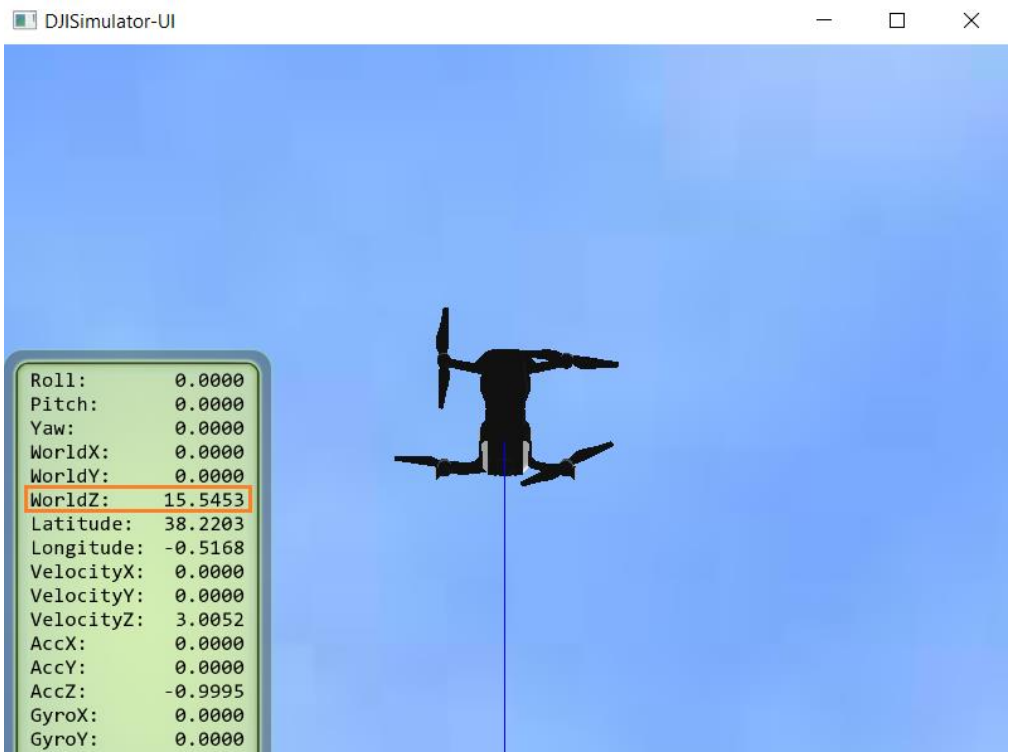


Ilustración 146. Ascenso 15m Demostración

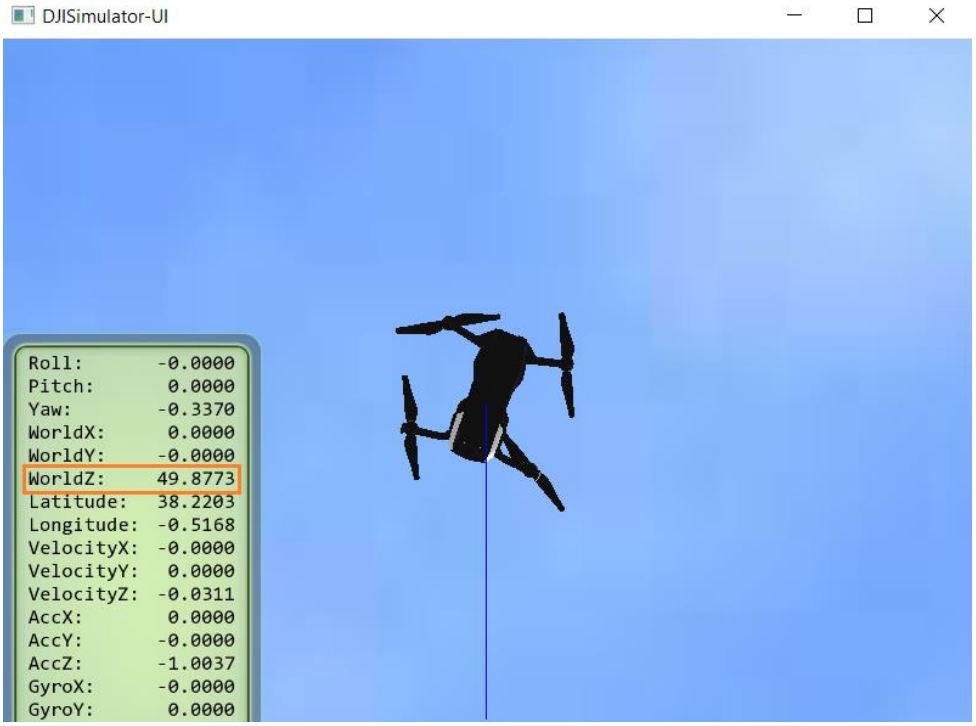


Ilustración 147. Ascenso 50m Demostración

Tal y como se puede comprobar en la imagen superior el dron no asciende hasta los 50 metros exactos, debido a que siempre existirá un pequeño margen de error propio del vehículo, en aspectos como la altura de vuelo o la colocación exacta sobre cada punto GPS indicado. Así se ha venido reiterando en distintas ocasiones, la necesidad de separar a los drones en altura y en distancia, para evitar que estos pequeños errores causados por los instrumentos del dron puedan ocasionar alguna colisión.

Cuando el aparato ha alcanzado la altura de vuelo (50m) comenzará a dirigirse al primer Waypoint. Dicho desplazamiento podremos comprobarlo a través de la ventana del simulador DJI y mediante el uso de la aplicación Android desarrollada, ya que el dron a medida que se va desplazando en el espacio así aparece representado sobre el mapa integrado en la aplicación, tal y como se muestra en la siguiente imagen.

Ilustración 148. Hacia el primer Waypoint Demostración

Una vez alcanzado el primer punto GPS, el dron se detendrá automáticamente tal y como se puede comprobar en las siguientes imágenes:

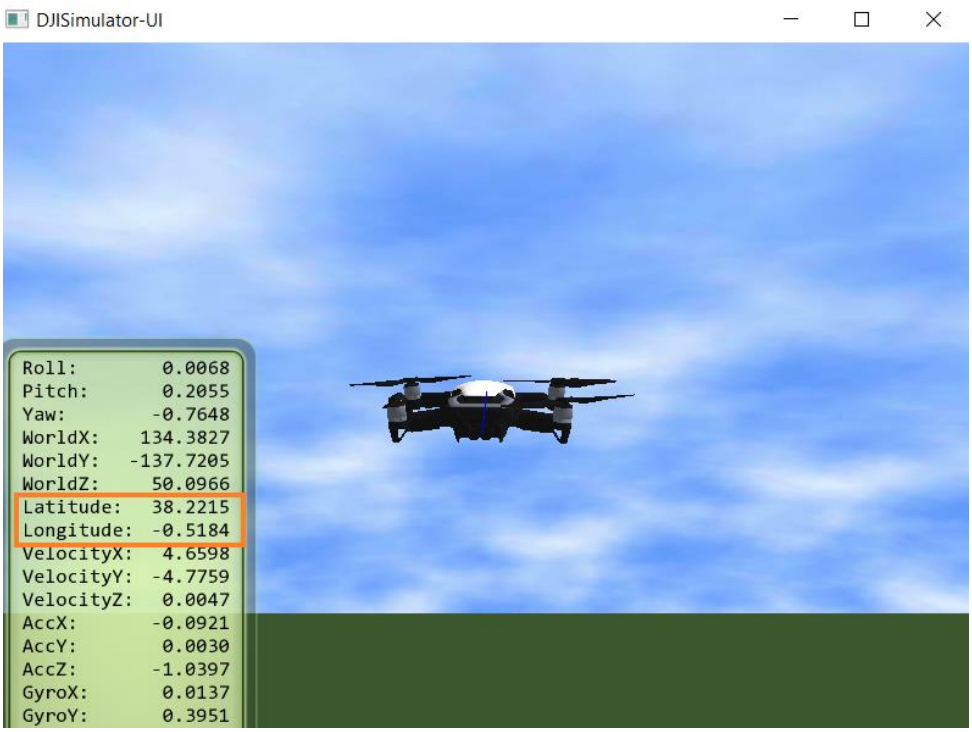


Ilustración 149. Primer Waypoint Simulador Demostración

Tal y como se puede comprobar en la imagen superior, el simulador empleado no informa con toda la precisión necesaria, acerca de algunos valores como son la latitud y la longitud sobre las que se encuentra el vehículo en cada momento. El programa se limita a ofrecer un número con solo 4 dígitos decimales, lo que en muchos casos puede resultar insuficiente. Por ello, el simulador durante la

ejecución va generando una serie de 'logs' que nos permite comprobar esos datos sin ningún tipo de redondeo o truncamiento.

Los ficheros generados los podremos descargar del UAV a través de la herramienta DJI Assistant, que venimos empleando.

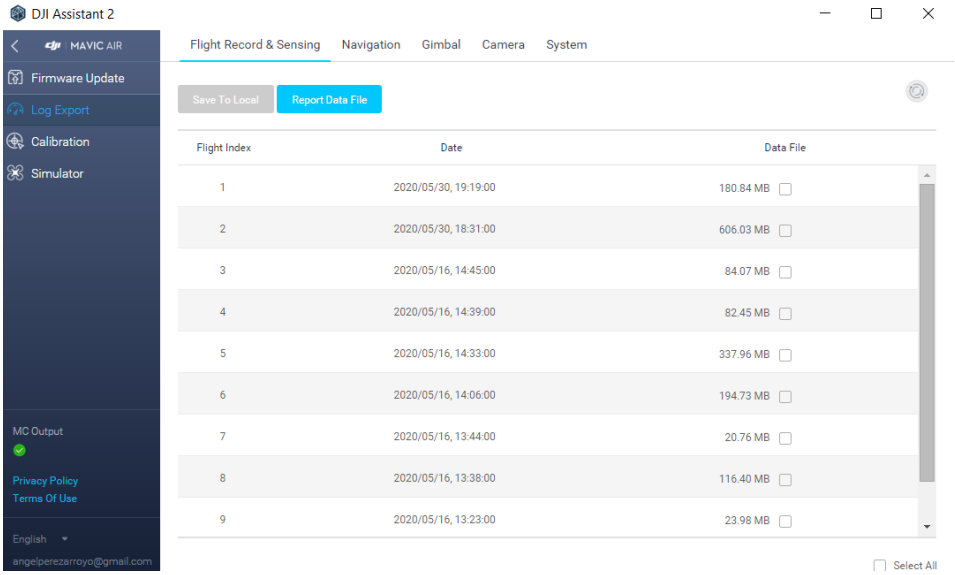


Ilustración 150. Logs DJI Assistant

En la imagen superior el dron se detiene justo sobre la primera coordenada GPS a la que debía dirigirse.

Además, en la imagen inferior podremos comprobar sobre el mapa de Google Maps, como el dron se ha detenido exactamente sobre el primer Waypoint indicado.

Ilustración 151. Primer Waypoint Mapa Demostración

Cuando el dron alcanza cualquier Waypoint, se detiene y realiza una fotografía. Para ello, tal y como se puede apreciar en las imágenes inferiores el dron mueve la cámara hacía abajo y dispara una fotografía. Finalmente, vuelve a colocar la cámara mirando al frente.

Ilustración 152. Cámara al frente Demostración

Ilustración 153. Cámara al abajo Demostración

Ilustración 154. Cámara al frente 2 Demostración

Cuando el dron ha completado la toma de la fotografía, éste se alineará con el siguiente punto GPS al que tiene que dirigirse para avanzar siempre de frente hacia el Waypoint, tal y como se puede apreciar en la siguiente imagen.

Ilustración 155. Dirección Waypoint 2 Demostración

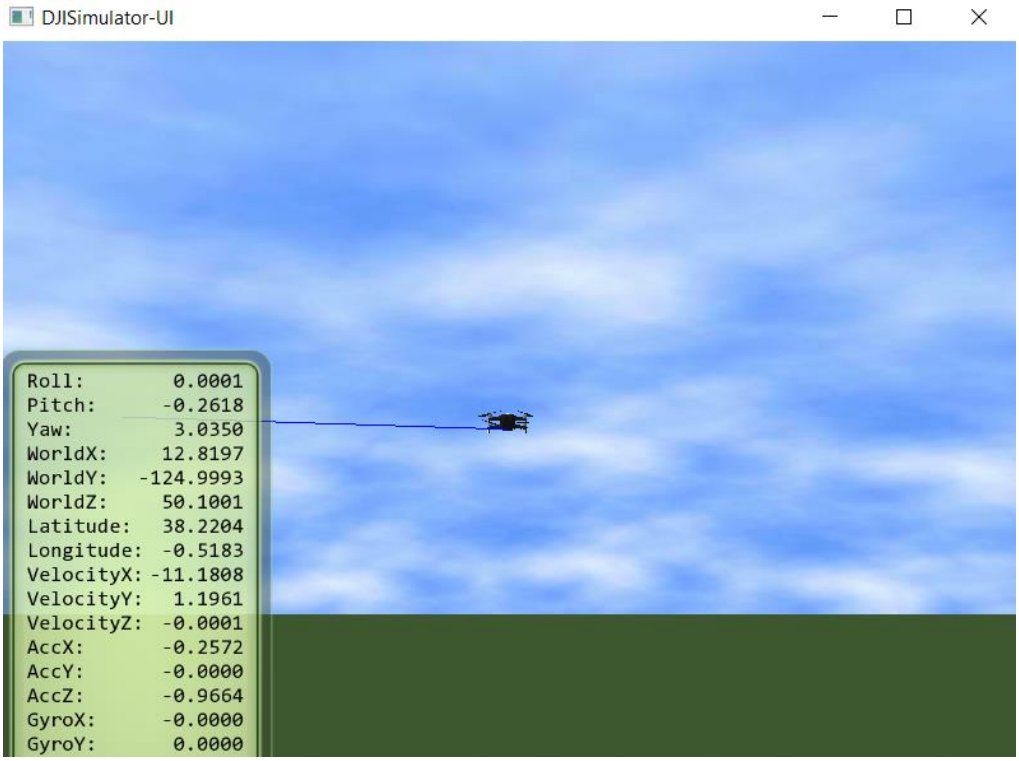


Ilustración 156. Dirección Waypoint 2 Simulador Demostración

Nuevamente, al alcanzar un Waypoint el dron se volverá a detener, bajará su cámara hacia abajo y disparará una fotografía, tal y como se ha realizado en el primer Waypoint de la misión.

Tal y como se puede apreciar en la imagen inferior, el aparato antes de llegar al punto GPS indicado inicia un proceso de desaceleración para detenerse totalmente sobre el Waypoint indicado al inicio.

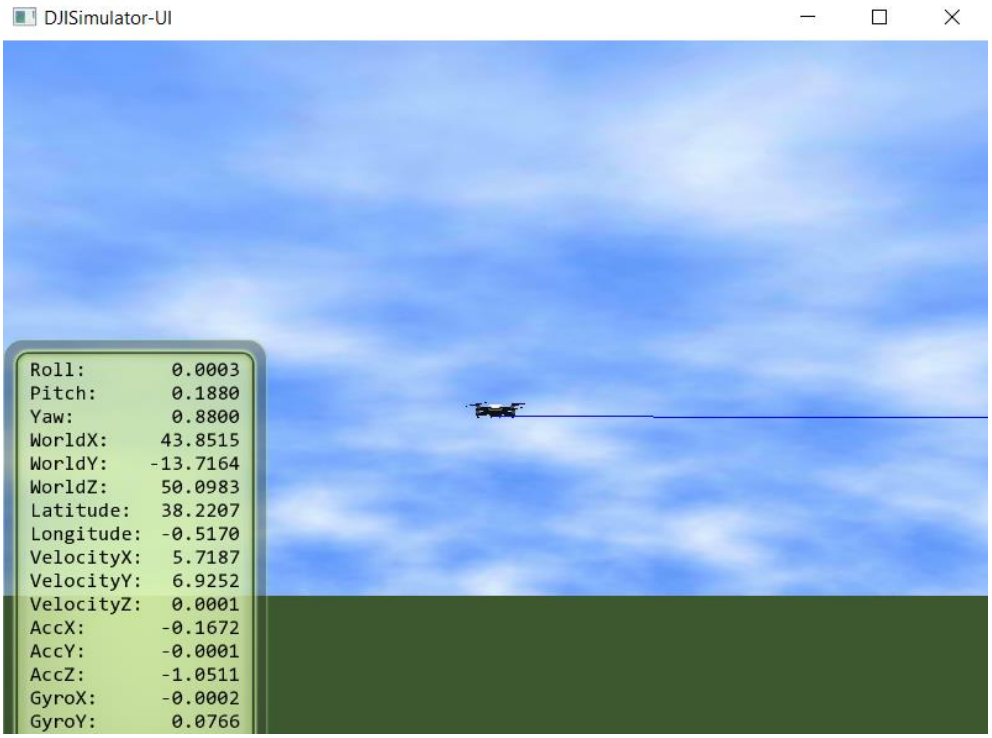


Ilustración 157. Frenando Waypoint 2 Demostración

Una vez el dron ha alcanzado el segundo punto GPS, la aeronave procederá a moverse hacia el tercero y último punto de la misión definida, tal y como se puede apreciar en la siguiente imagen.

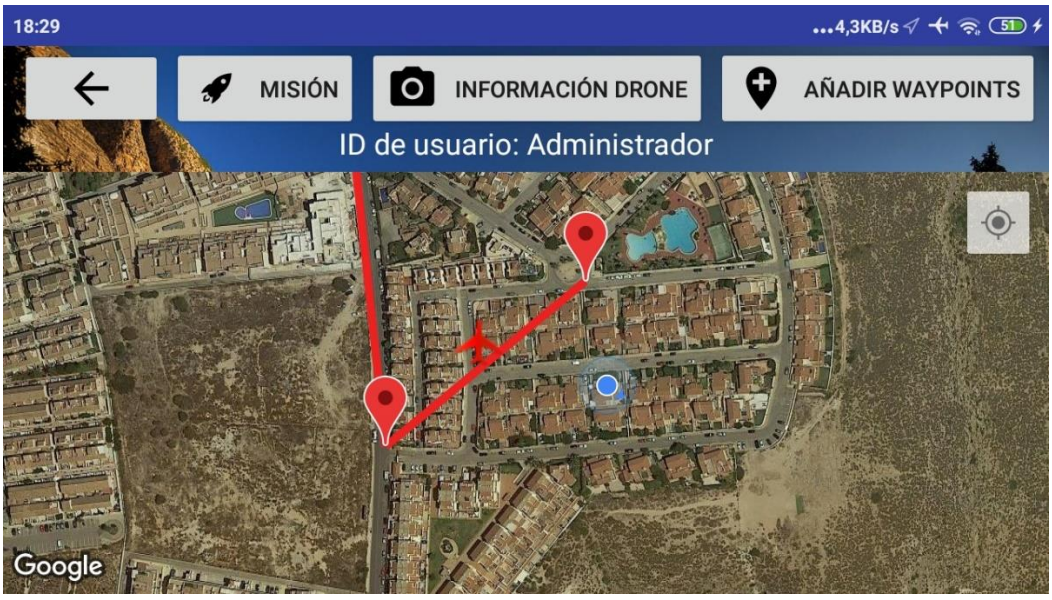


Ilustración 158. Hacia el 3º Waypoint Demostración

Alcanzado el último Waypoint de la misión, la aeronave tomará la fotografía correspondiente tal y como se cargó en la memoria del dron.

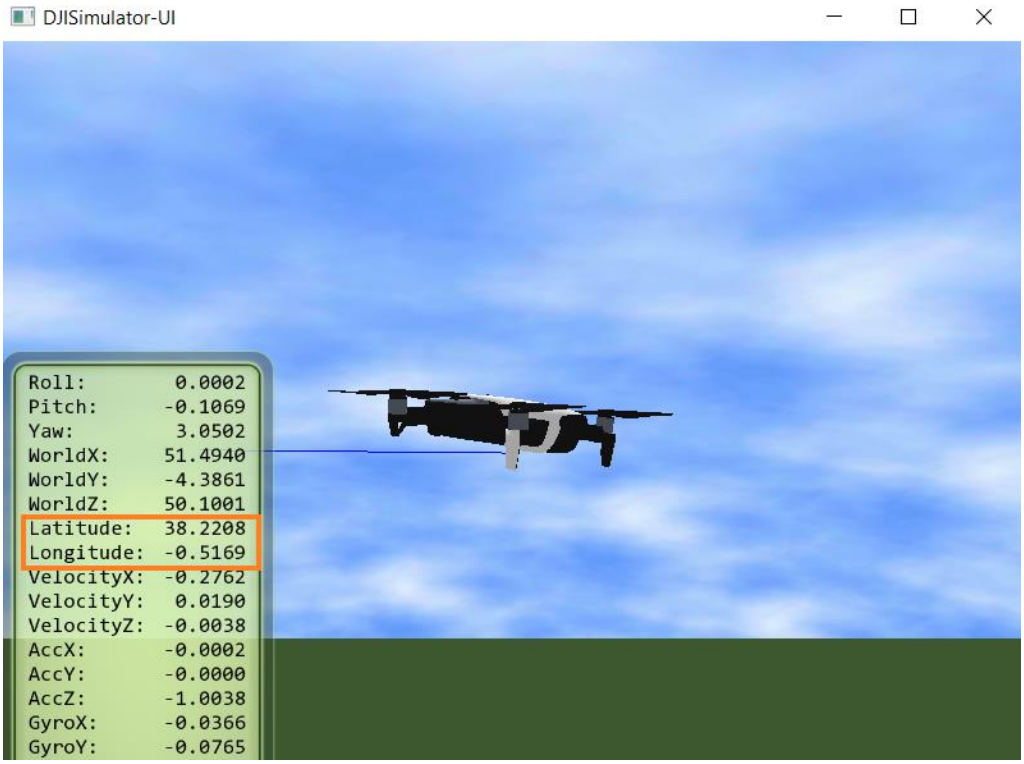


Ilustración 159. Waypoint 3 Demostración

Finalizadas las acciones a realizar el dron volverá de forma automática al punto de despegue para iniciar el proceso de aterrizaje.

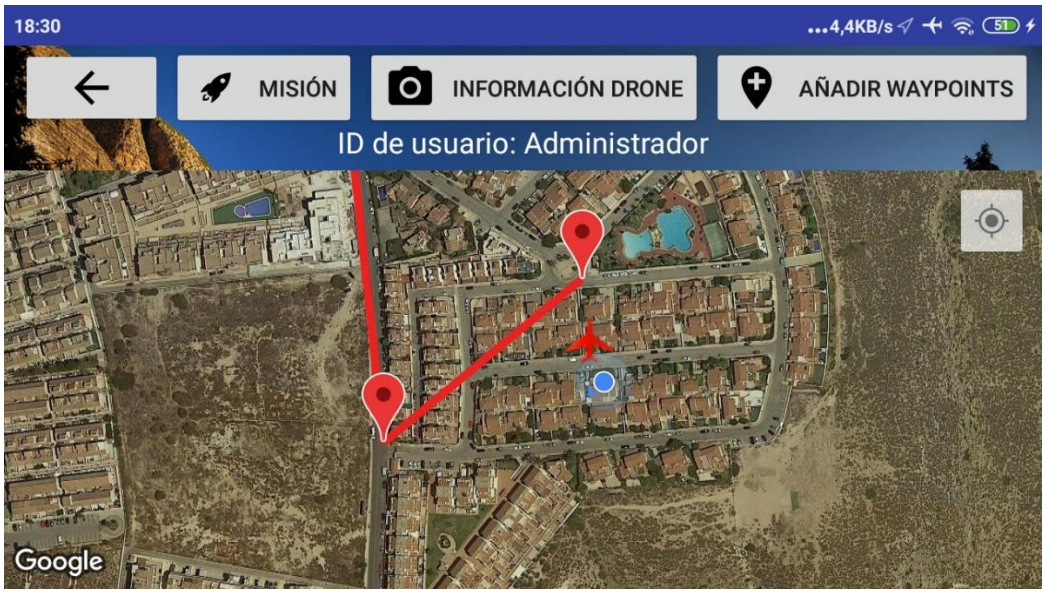


Ilustración 160. Vuelta a casa Demostración

En ese punto el dron comenzará a descender hasta una altura de 0.5 metros del suelo como se indica en las siguientes imágenes.

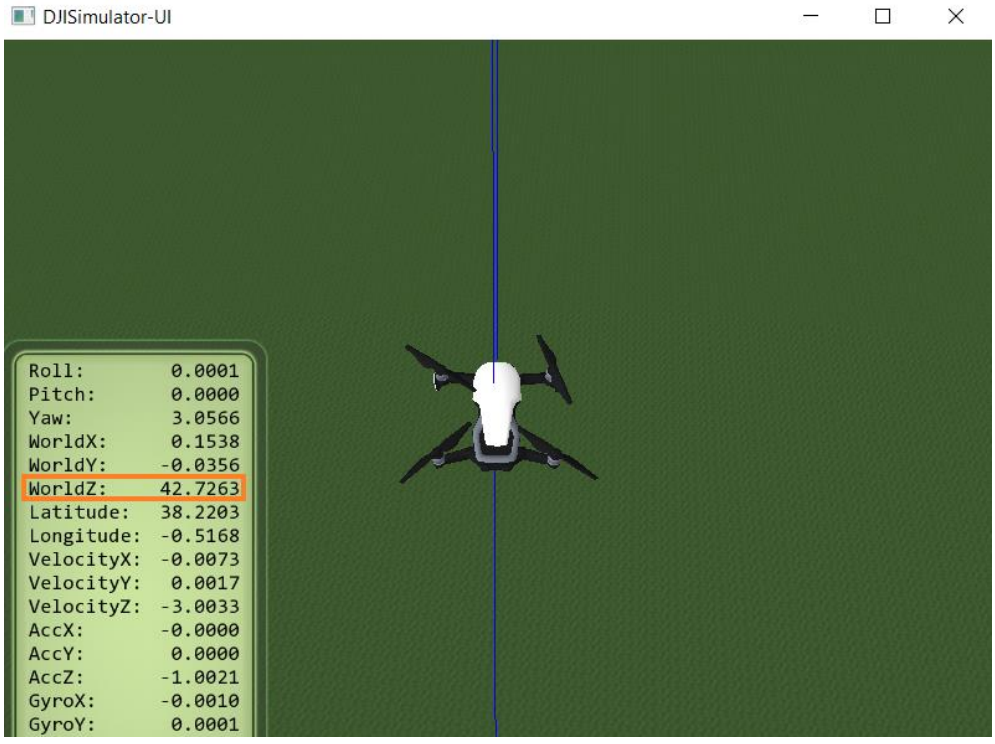


Ilustración 161. Descenso (42m) Demostración

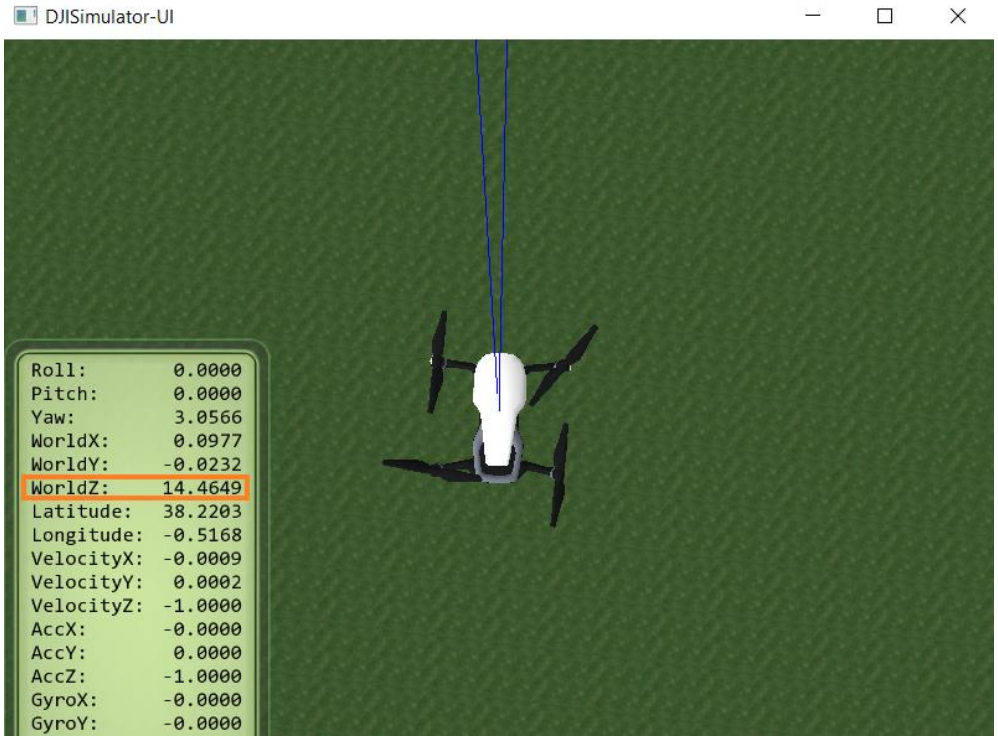


Ilustración 162. Descenso (14m) Demostración

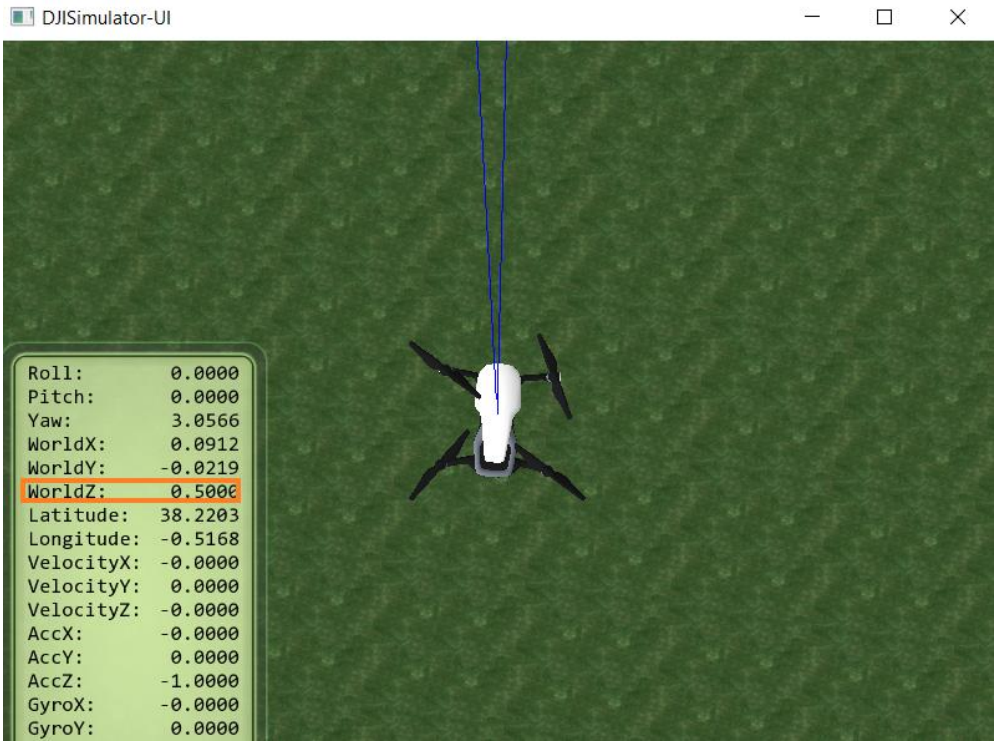


Ilustración 163. Descenso (0.5m) Demostración

Cuando el dron haya alcanzado esa altura entrará en un estado de espera hasta la confirmación de aterrizaje por parte del usuario. Para ello, simplemente mantendremos 2 o 3 segundos el 'stick' que gestiona el ascenso o descenso de la aeronave hacia abajo, pasados esos segundos el dron descenderá hasta el suelo de manera automática y apagará los motores, tal y como se puede ver en la imagen inferior.

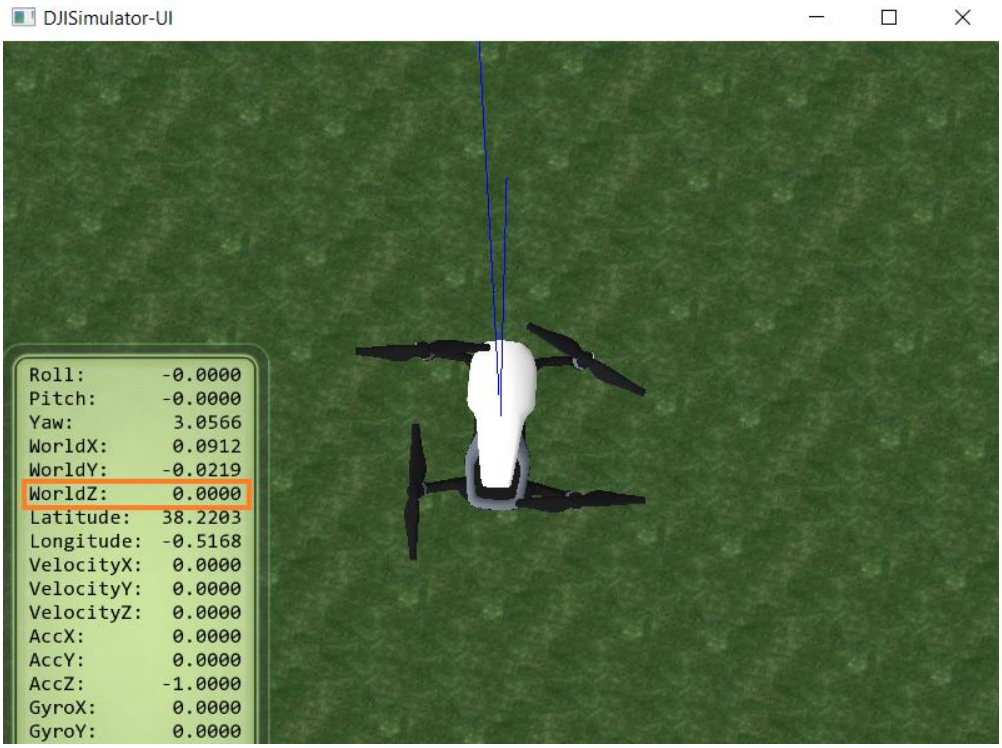


Ilustración 164. Descenso finalizado Demostración

Finalmente, procederemos a realizar una misión similar a la explicada anteriormente, pero para demostrar la posibilidad de ejecutar misiones con drones DJI de manera simultánea y controlada desde un único terminal, se procederá a realizar otro ejemplo, en el que se emplearán dos terminales Android.

Para ello, deberemos registrar un nuevo dron dentro de la misión utilizada con anterioridad (misión con ID 2). A diferencia del ejemplo explicado, éste nuevo usuario no será administrador ya que deseamos simular un comportamiento, como el que se llevaría a cabo en la vida real.

Por lo tanto, una vez registrado el nuevo usuario, podremos comprobar que en la tabla 'drones' existen dos usuarios registrados para la misión con ID 2, tal y como se puede verificar en la imagen inferior.

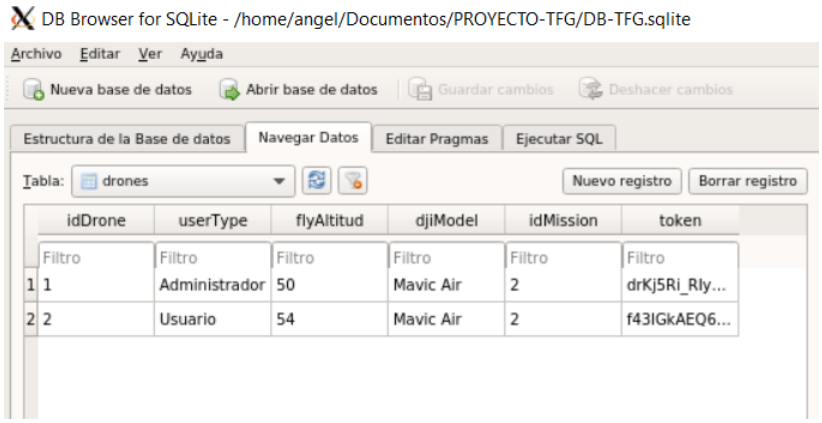


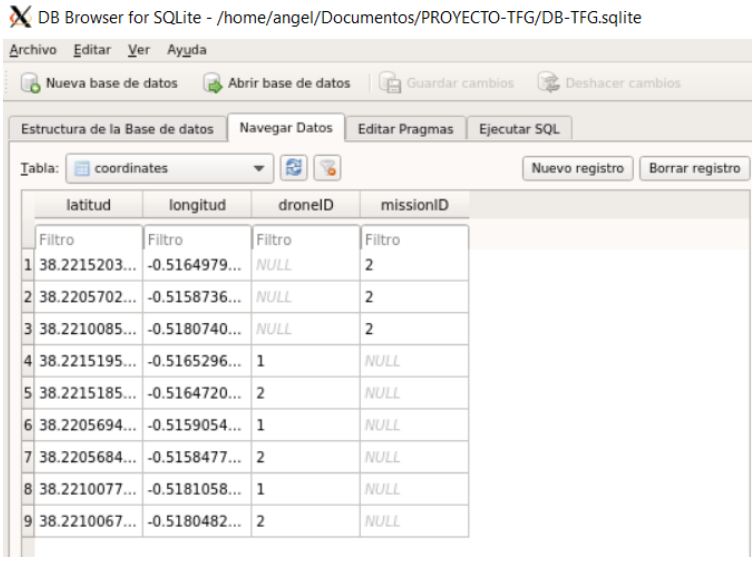
Ilustración 165. 2 usuarios registrados Demostración

Seguidamente, desde el terminal que hará las funciones de Administrador de la misión, se procederá a definir unos nuevos Waypoints o puntos GPS, tal y como se puede ver en la imagen inferior.

Ilustración 166. Administrador Demostración Final

Una vez definidos dichos Waypoints se procederá a su envío al servidor para que éste calcule los puntos relativos para cada uno de los drones, que van a componer la misión.

Una vez el algoritmo encargado del cálculo de estas coordenadas relativas se ha completado, podremos verificar como en la tabla “coordinates” ya se disponen de los puntos GPS para cada uno de los drones.



	latitud	longitud	droneID	missionID
	Filtro	Filtro	Filtro	Filtro
1	38.2215203...	-0.5164979...	NULL	2
2	38.2205702...	-0.5158736...	NULL	2
3	38.2210085...	-0.5180740...	NULL	2
4	38.2215195...	-0.5165296...	1	NULL
5	38.2215185...	-0.5164720...	2	NULL
6	38.2205694...	-0.5159054...	1	NULL
7	38.2205684...	-0.5158477...	2	NULL
8	38.2210077...	-0.5181058...	1	NULL
9	38.2210067...	-0.5180482...	2	NULL

Ilustración 167. Cálculo coordenadas Demostración Final

Seguidamente, podremos descargar los Waypoints para cada dron de la misión en el terminal asociado, tal como aparece en las siguientes imágenes.

Ilustración 168. Waypoints Administrador Demostración Final

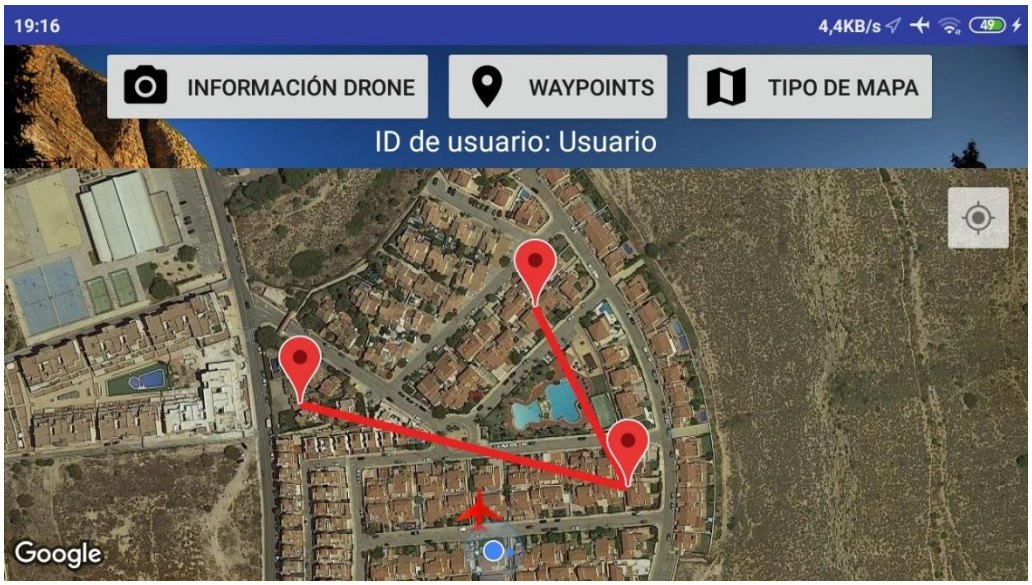


Ilustración 169. Waypoints Usuario Demostración Final

Cuando ambos dispositivos dispongan de sus correspondientes puntos GPS, podremos iniciar la misión desde el terminal Administrador. Pulsando el botón para iniciar la misión, ambos terminales recibirán la orden correspondiente, pero debido a las limitaciones de poseer un único dron, tan sólo el terminal conectado al vehículo iniciará el proceso de transferencia de datos.

Así, se puede deducir que la aplicación realizada tiene un diseño escalable, ya que puede funcionar con un gran número de UAVs al mismo tiempo, ofreciendo a los usuarios una gran cantidad de posibles aplicaciones.

Ilustración 170. Waypoint 1 Demostración Final

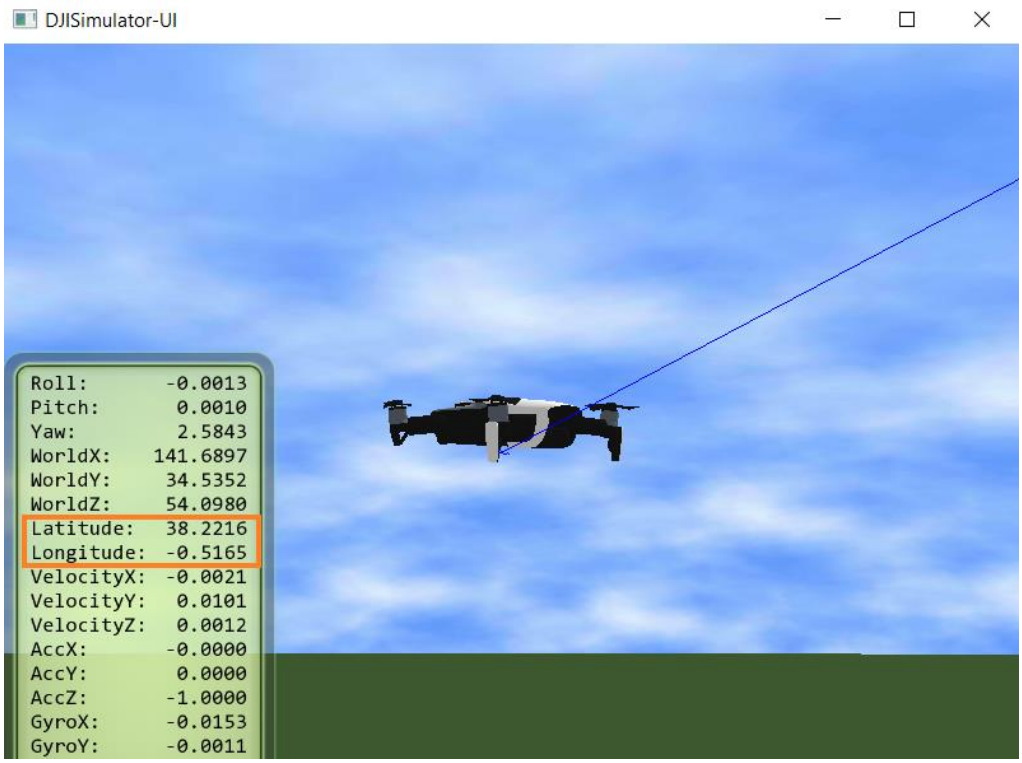


Ilustración 171. Waypoint 1 Simulador Demostración Final

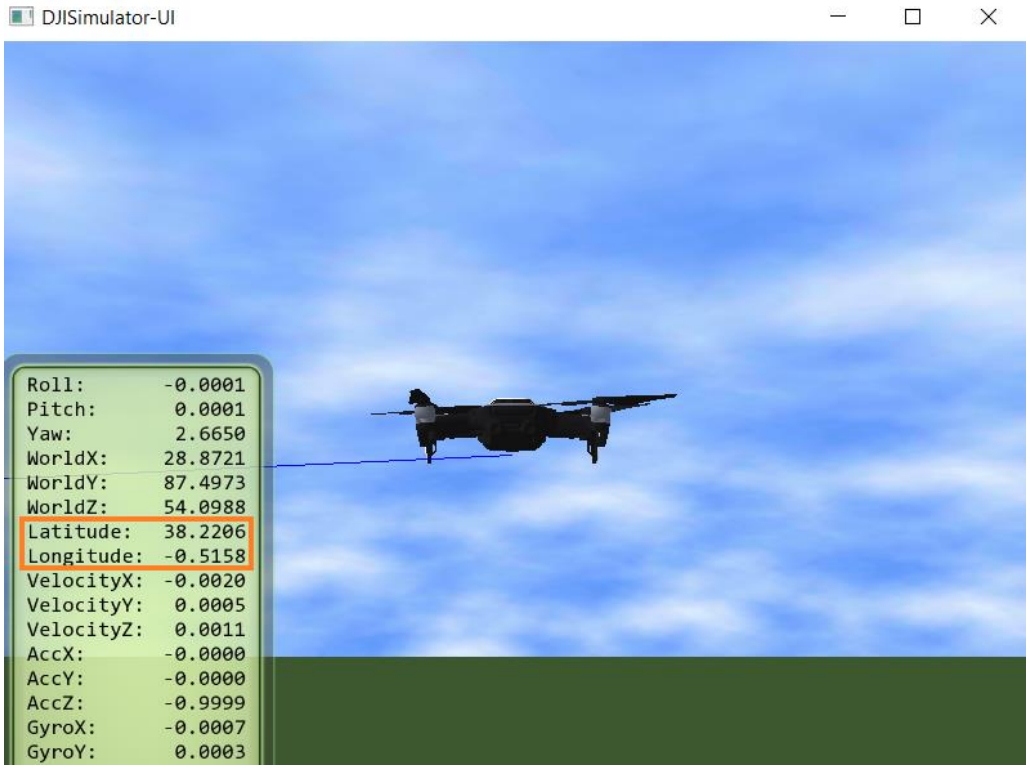


Ilustración 172. Waypoint 2 Simulador Demostración Final

Ilustración 173. Hacia Waypoint 3 Demostración Final

Ilustración 174. Vuelta a punto de despegue Demostración Final

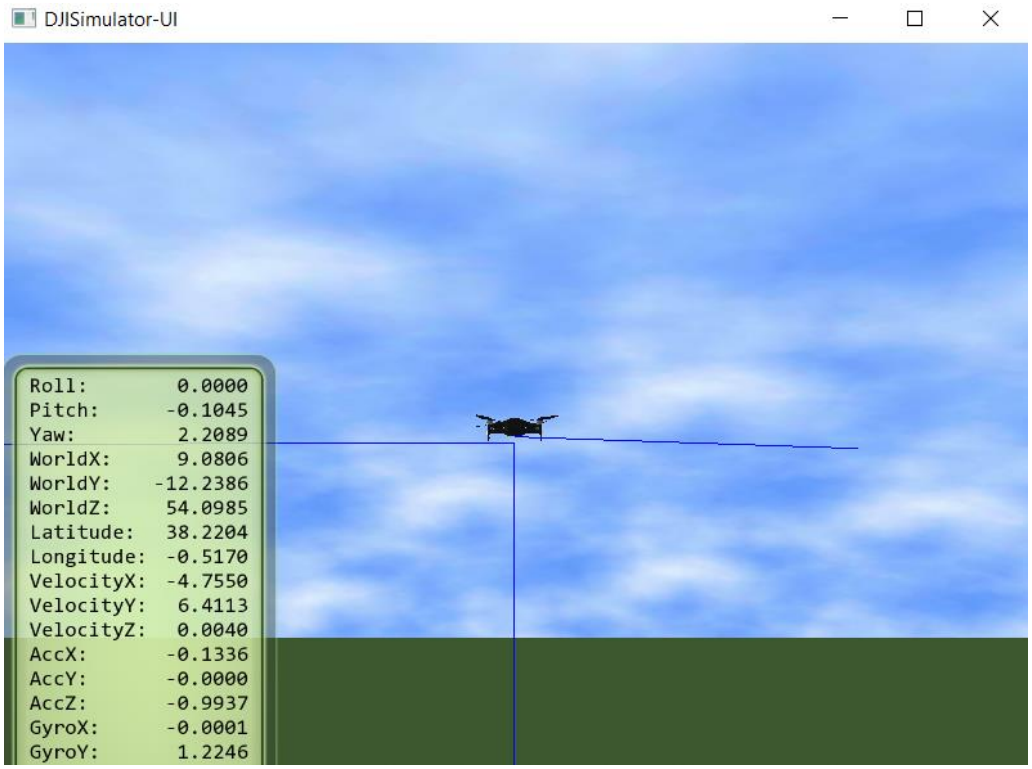


Ilustración 175. Vuelta a punto de despegue Simulador Demostración Final

4. Abreviaturas utilizadas

- UAV[1]: Aeronave no tripulada
- UAVs[2]: Aeronaves no tripuladas

5. Apéndice

5.1. Apéndice A. Tutoriales

A lo largo de este primer apéndice de la memoria Final de Grado se procederá a realizar tres tutoriales básicos. En ellos se explicará de una manera más detallada factores fundamentales para el desarrollo del proyecto realizado y que, por lo tanto, pretenden servir como guía para desarrolladores, que vayan a implementar aplicaciones y/o funcionalidades a las descritas a lo largo de esta memoria.

Entre los conceptos a exponer durante estos tutoriales tendremos los siguientes: incluir el SDK de DJI en un proyecto de Android Studio, el uso de la API de Google Maps para Android y la integración de Firebase tanto en el cliente como en servidor.

5.1.1. Integración del SDK de DJI en Android Studio

A lo largo del presente tutorial se explicará de manera detallada cada uno de los pasos necesarios para poder integrar y utilizar el Mobile SDK dentro de un proyecto en Android Studio.

En primer lugar, para poder comprobar el funcionamiento de la aplicación a implementar, resultará estrictamente necesario disponer de un dron de la marca DJI, que se encuentre dentro de la lista de productos soportados. Esto será necesario ya que, para poder ejecutar simulaciones de cualquier tipo utilizando el software creado para ello, necesitaremos conectar el dron al ordenador.

Además del dron necesitaremos un dispositivo Android con una versión 4.4 o superior, el SDK de DJI necesita esta versión como mínimo para funcionar de manera correcta. Por último, tan sólo será necesario disponer de un ordenador con sistema operativo Linux, Windows o macOS y de una conexión a internet. En caso de cumplir con todos los requisitos previos podremos proceder con el tutorial.

El primer paso para poder utilizar el SDK de DJI será crear una cuenta como desarrolladores, algo que no debería llevar más de cinco minutos. Enlace registro usuario.

Una vez creada la cuenta, el siguiente paso será descargar Android Studio, si no lo tenemos. Una vez descargado e instalado, no será necesario crear ningún dispositivo virtual, ya que el SDK de DJI tan sólo funciona en dispositivos físicos. Después instalaremos las siguientes herramientas: "Google USB Driver", "Google Play Services", "Android SDK Tools", "NDK (Side by side)" y "Google Play APK Expansion library". Para ello nos dirigimos a 'Herramientas' -> 'SDK Manager' -> 'SDK Tools'. Una vez hayamos alcanzado la opción de 'SDK Tools', veremos un listado de herramientas adicionales para añadir a Android Studio.

Cuando tengamos completado el paso anterior ya podremos crear un nuevo proyecto en Android Studio. En cuanto a la "MainActivity" la escogeremos de tipo "Empty" y con la versión mínima de Android 4.4 o API 19, ya que como se ha comentado anteriormente necesitaremos esa versión como mínimo para integrar correctamente el SDK.

Después debemos dirigirnos a la página de DJI Developer para generar una clave y añadirla a nuestro proyecto de Android Studio. Procederemos a iniciar sesión. Es fundamental integrar la clave de forma correcta, ya que de no hacerlo la aplicación se cerrará inmediatamente después de abrirla. Una vez hayamos iniciado sesión, nos dirigiremos al apartado de 'Apps'. Dentro de esta opción encontraremos el botón "CREATE APP", que pulsemos, entonces se nos solicitarán datos de nuestro proyecto de Android Studio como el nombre del paquete o el nombre de la aplicación.

Una vez rellenados todos los datos, recibiremos un correo de confirmación a nuestro e-mail. Cuando confirmemos la creación de la aplicación, podremos volver a la misma página en la que nos encontrábamos previamente y copiar la clave generada.

Al disponer de la clave para nuestra aplicación en Android, ya podremos integrar el SDK de DJI porque tendremos todo lo necesario para ello.

El primer paso por realizar será la modificación del archivo "AndroidManifest.xml" al que añadiremos lo siguiente:

1. Como primer hijo del elemento 'manifest' incluiremos los siguientes permisos necesarios para que la aplicación funcione correctamente.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<uses-permission android:name="android.permission.CHANGE_CONFIGURATION" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
```

En cuanto a los tres errores que salen al final, los descartaremos ya que esto es totalmente normal y la aplicación funcionará correctamente de igual manera.

2. A continuación, añadiremos el contenido de la imagen para poder acceder a cualquier accesorio conectado al USB del terminal.

```
<uses-feature
    android:name="android.hardware.usb.accessory"
    android:required="true" />
<uses-feature
    android:name="android.hardware.usb.host"
    android:required="false" />
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true" />
```

3. Dentro de la etiqueta ‘application’ añadiremos un nuevo campo llamado “android:name”, al que le daremos el valor de “.MApplication”. Ésta será una clase que crearemos posteriormente para inicializar el SDK, y aparecerá un error que se solucionará posteriormente.

4. Después debemos añadir lo expresado en la siguiente imagen como primer hijo del elemento ‘application’.

```
<!-- DJI SDK -->
<uses-library android:name="com.android.future.usb.accessory" />
<uses-library android:name="org.apache.http.legacy" android:required="false" />
<meta-data
    android:name="com.dji.sdk.API_KEY"
    android:value="" />
<activity
    android:name="dji.sdk.sdkmanager.DJIAoaControllerActivity"
    android:theme="@android:style/Theme.Translucent" >
    <intent-filter>
        <action android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
    </intent-filter>
    <meta-data
        android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
        android:resource="@xml/accessory_filter" />
</activity>

<!-- DJI SDK -->
```

Será dentro del campo “android:value” donde debemos insertar la clave obtenida previamente.

5. La última modificación, que debemos realizar sobre este documento, se efectuará al final de este. Para ello debemos modificar el elemento ‘activity’ de la siguiente manera:

```
        <activity android:name=".MainActivity"
            android:configChanges="orientation"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

6. El siguiente paso será la modificación del fichero ‘build.gradle (Module:app)’cambiando el contenido de los campos ‘defaultConfig’ y ‘buildTypes’ a la configuración que se muestra a continuación:

```
defaultConfig {
    applicationId "com.tfg.importsdkdemo"
    minSdkVersion 19
    targetSdkVersion 29
    versionCode 1
    versionName "1.0"

    testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    ndk {
        // On x86 devices that run Android API 23 or above, if the application is targeted with API 23 or
        // above, FFmpeg lib might lead to runtime crashes or warnings.
        abiFilters 'armeabi-v7a', 'x86', 'arm64-v8a'
    }
}

buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
    }
}
```

Por último, destacar que, el nombre de la aplicación que aparecerá será el que se haya utilizado previamente y el 'targetSdkVersion' también podrá variar en función de la última versión disponible de Android.

7. A continuación, y para evitar errores de funcionamiento de la aplicación durante su ejecución debemos añadir los siguientes elementos como hijos del elemento 'android'.

```
dexOptions {
    javaMaxHeapSize "4g"
}

packagingOptions{
    doNotStrip "*/*/libdjivideo.so"
    doNotStrip "*/*/libSDKRelativeJNI.so"
    doNotStrip "*/*/libFlyForbid.so"
    doNotStrip "*/*/libduml_vision_bokeh.so"
    doNotStrip "*/*/libyuv2.so"
    doNotStrip "*/*/libGroudStation.so"
    doNotStrip "*/*/libFRCorkscrew.so"
    doNotStrip "*/*/libUpgradeVerify.so"
    doNotStrip "*/*/libFR.so"
    doNotStrip "*/*/libDJIFlySafeCore.so"
    doNotStrip "*/*/libdjifs_jni.so"
    doNotStrip "*/*/libsfjni.so"
    exclude 'META-INF/rxjava.properties'
}

compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}
```

8. Por último, dentro del campo de dependencias que se encuentra al final del fichero añadiremos lo siguiente:

```
implementation('com.dji:dji-sdk:4.11.2', {
    /**
     * Uncomment the "library-anti-distortion" if your app does not need Anti Distortion
     * for Mavic 2 Pro and Mavic 2 Zoom.
     * Uncomment the "fly-safe-database" if you need database for release, or we will
     * download it when DJISDKManager.getInstance().registerApp
     * is called.
     * Both will greatly reducing the size of the APK.
     */
    exclude module: 'library-anti-distortion'
    exclude module: 'fly-safe-database'
})
compileOnly 'com.dji:dji-sdk-provided:4.11.2'
```

9. La siguiente y última modificación la debemos realizar sobre el fichero 'build.gradle (Project: nombreProjecto)'. En este fichero modificaremos la versión de Gradle al número 3.2.1, ya que si se utiliza alguna versión posterior la aplicación se detendrá al ejecutarla.

10. Por último, podremos sincronizar el proyecto, para que Android Studio descargue todos los paquetes adicionales que vayamos a necesitar.

11. Si hemos llegado hasta este punto, tan sólo tendremos que aplicar unas pocas líneas de programación en Java y ya nos encontraremos en disposición de utilizar cualquier funcionalidad del DJI Mobile SDK. El siguiente paso, será crear una clase llamada "MApplication" dentro de la cual deberemos implementar el siguiente método:

```
@Override
protected void attachBaseContext(Context paramContext)
{
    super.attachBaseContext(paramContext);
    Helper.install( app: MApplication.this);
}
```

12. Como último paso, realizaremos la siguiente implementación en la clase "MainActivity".

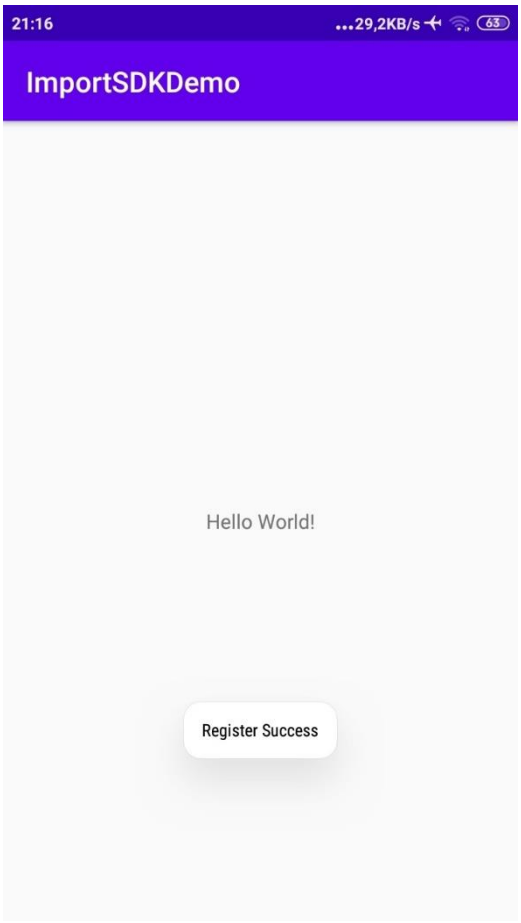
```java
public class MainActivity extends AppCompatActivity
{
    private static final String TAG = MainActivity.class.getName();
    public static final String FLAG_CONNECTION_CHANGE = "dji_sdk_connection_change";
    private static BaseProduct mProduct;
    private Handler mHandler;

    private static final String[] REQUIRED_PERMISSION_LIST = new String[]{
            Manifest.permission.VIBRATE,
            Manifest.permission.INTERNET,
            Manifest.permission.ACCESS_WIFI_STATE,
            Manifest.permission.WAKE_LOCK,
            Manifest.permission.ACCESS_COARSE_LOCATION,
            Manifest.permission.ACCESS_NETWORK_STATE,
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.CHANGE_WIFI_STATE,
            Manifest.permission.WRITE_EXTERNAL_STORAGE,
            Manifest.permission.BLUETOOTH,
            Manifest.permission.BLUETOOTH_ADMIN,
            Manifest.permission.READ_EXTERNAL_STORAGE,
            Manifest.permission.READ_PHONE_STATE,
    };
    private List<String> missingPermission = new ArrayList<>();
    private AtomicBoolean isRegistrationInProgress = new AtomicBoolean( initialValue: false);
    private static final int REQUEST_PERMISSION_CODE = 12345;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        // When the compile and target version is higher than 22, please request the following
        // permission at runtime to ensure the SDK works well.
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M)
            checkAndRequestPermissions();

        setContentView(R.layout.activity_main);

        //Initialize DJI SDK Manager
        mHandler = new Handler(Looper.getMainLooper());
    }

    /**
     * Checks if there is any missing permissions, and
     * requests runtime permission if needed.
     */
    private void checkAndRequestPermissions()
    {
        // Check for permissions
        for (String eachPermission : REQUIRED_PERMISSION_LIST)
        {
            if (ContextCompat.checkSelfPermission( context: this, eachPermission)
                    != PackageManager.PERMISSION_GRANTED)
                missingPermission.add(eachPermission);
        }
```

```
    // Request for missing permissions
    if (missingPermission.isEmpty())
        startSDKRegistration();

    else if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M)
    {
        showToast( toastMsg: "Need to grant the permissions!");
        ActivityCompat.requestPermissions( activity: this,
                missingPermission.toArray(new String[missingPermission.size()]),
                REQUEST_PERMISSION_CODE);
    }
}

/**
 * Result of runtime permission request
 */
@Override
public void onRequestPermissionsResult(int requestCode,
                                       @NonNull String[] permissions,
                                       @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    // Check for granted permission and remove from missing list
    if (requestCode == REQUEST_PERMISSION_CODE)
    {
        for (int i = grantResults.length - 1; i >= 0; i--)
        {
            if (grantResults[i] == PackageManager.PERMISSION_GRANTED)
                missingPermission.remove(permissions[i]);
        }
    }

    // If there is enough permission, we will start the registration
    if (missingPermission.isEmpty())
        startSDKRegistration();

    else
        showToast( toastMsg: "Missing permissions!!!");
}

private void startSDKRegistration()
{
    if (isRegistrationInProgress.compareAndSet( expect: false,  update: true))
    {
        AsyncTask.execute(new Runnable()
        {
            @Override
            public void run() {
                showToast( toastMsg: "registering, pls wait...");

                DJISDKManager.getInstance().registerApp(MainActivity.t
                        his.getApplicationContext(), new DJISDKManager.SDKManagerCallback()
```

```
@Override
public void onRegister(DJIError djiError)
{
    if (djiError == DJISDKError.REGISTRATION_SUCCESS)
    {
        showToast( toastMsg: "Register Success");
        DJISDKManager.getInstance().startConnectionToProduct();
    }

    else
        showToast( toastMsg: "Register sdk fails, please check " +
                "the bundle id and network connection!");

    Log.v(TAG, djiError.getDescription());
}

@Override
public void onProductDisconnect()
{
    Log.d(TAG,  msg: "onProductDisconnect");
    showToast( toastMsg: "Product Disconnected");
    notifyStatusChange();
}

@Override
public void onProductConnect(BaseProduct baseProduct)
{
    Log.d(TAG, String.format("onProductConnect newProduct:%s", baseProduct));
    showToast( toastMsg: "Product Connected");
    notifyStatusChange();
}

@Override
public void onComponentChange(BaseProduct.ComponentKey componentKey,
                              BaseComponent oldComponent,
                              BaseComponent newComponent)
{
    if (newComponent != null)
    {
        newComponent.setComponentListener(new BaseComponent.ComponentListene
        {
            @Override
            public void onConnectivityChange(boolean isConnected)
            {
                Log.d(TAG,  msg: "onComponentConnectivityChanged: " +
                        isConnected);
                notifyStatusChange();
            }
        });
    }

    Log.d(TAG,
            String.format("onComponentChange key:%s, oldComponent:%s, " +
                          "newComponent:%s",
                    componentKey,
                    oldComponent,
                    newComponent));
}
```

```
                    @Override
                    public void onInitProcess(DJISDKInitEvent djisdkInitEvent, int i)
                    {
                    }

                    @Override
                    public void onDatabaseDownloadProgress(long l, long l1)
                    {
                    }

                });
            }
        });
    }
}

private void notifyStatusChange()
{
    mHandler.removeCallbacks(updateRunnable);
    mHandler.postDelayed(updateRunnable, delayMillis: 500);
}

private Runnable updateRunnable = new Runnable()
{
    @Override
    public void run()
    {
        Intent intent = new Intent(FLAG_CONNECTION_CHANGE);
        sendBroadcast(intent);
    }
};

private void showToast(final String toastMsg)
{
    Handler handler = new Handler(Looper.getMainLooper());
    handler.post(new Runnable()
    {
        @Override
        public void run() {
            Toast.makeText(getApplicationContext(), toastMsg, Toast.LENGTH_LONG).show();
        }
    });
}
}
```

12. Después comprobaremos la aplicación y verificaremos que ésta se registra correctamente. Si obtenemos un resultado como el que se muestra en la imagen inferior, el proceso habrá finalizado, al haber completado correctamente los pasos previos.



Por último, destacar que toda esta información ha sido obtenida de la documentación de DJI.

5.1.2. Integración y uso del SDK de Google Maps para Android.

Para la realización de este tutorial que consiste en conocer cómo integrar el Google Maps SDK para Android, serán necesarias las siguientes herramientas: Android Studio, disponer de una cuenta de Google y un terminal Android. El dispositivo Android puede ser tanto físico como virtual.

El primer paso será dirigirnos a Google Cloud Platform. En ese sitio web, encontraremos la opción "Consola" en la parte superior derecha, sobre la que haremos 'click'. Cuando hayamos entrado a la consola de Google Cloud Platform debemos crear un nuevo proyecto. Después desplegaremos el menú de navegación (botón localizado esquina superior izquierda) y nos dirigiremos a la opción llamada "APIs y Servicios". Seguidamente, haremos 'click' sobre la opción llamada "HABILITAR APIs Y SERVICIOS". Para este tutorial necesitaremos habilitar las siguientes APIs: Maps SDK for Android y Places API.

A continuación, volveremos al panel de control de APIs Y SERVICIOS y encontraremos que en la parte izquierda de la pantalla hay una opción llamada "Credenciales" sobre la que haremos un 'click', al igual que sobre la opción "CREAR CREDENCIALES" -> "Clave de API".

Después tendremos disponible una clave de API, que podremos limitar para poder utilizar tan sólo desde ciertas aplicaciones. En cuanto a este paso decir, que es totalmente opcional por lo que se deja a elección del usuario el restringir el uso de la API o no.

A continuación, nos dirigiremos a Android Studio para generar un nuevo proyecto. En la realización de este tutorial se recomienda, generar un proyecto con la "MainActity" de tipo 'Empty' y utilizando una versión de la API de Android >= 19.

Cuando dispongamos de nuestro nuevo proyecto en Android Studio, iremos a nuestro archivo "AndroidManifest.xml". Una vez allí, añadiremos el siguiente permiso:

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

Además, debemos insertar como primer hijo dentro del elemento ‘application’ lo siguiente:

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="" />
```

Dentro del campo “android:value”, será donde insertamos la clave de API obtenida previamente en el Google Cloud Platform.

El siguiente paso será incluir las siguientes dos dependencias dentro del campo “dependecies” del fichero “build.gradle (Module: app)”:

implementation 'com.google.android.gms:play-services-maps:17.0.0'

implementation 'com.google.android.libraries.places:places:2.2.0'

Finalmente, tan sólo nos quedará por implementar alguna aplicación, que utilice alguna o ambas de estas APIs para comprobar que todos los procedimientos previos los hemos realizado correctamente. A continuación, se expone a modo de ejemplo un código con el que se podrá comprobar si los pasos previos se han realizado correctamente:

```
public class MapsActivity extends AppCompatActivity implements OnMapReadyCallback
{
    private static final String TAG = MapsActivity.class.getSimpleName();
    private GoogleMap mMap;
    private CameraPosition mCameraPosition;

    // The entry point to the Places API.
    private PlacesClient mPlacesClient;

    // The entry point to the Fused Location Provider.
    private FusedLocationProviderClient mFusedLocationProviderClient;

    private static final int DEFAULT_ZOOM = 15;
    private static final int PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION = 1;
    private boolean mLocationPermissionGranted;

    // The geographical location where the device is currently located. That is, the last-known
    // location retrieved by the Fused Location Provider.
    private Location mLastKnownLocation;

    private final LatLng mDefaultLocation = new LatLng( v: 38.195123,  v1: -0.559253);

    // Keys for storing activity state.
    private static final String KEY_CAMERA_POSITION = "camera_position";
    private static final String KEY_LOCATION = "location";
```

```
// Used for selecting the current place.
private static final int M_MAX_ENTRIES = 5;
private String[] mLikelyPlaceNames;
private String[] mLikelyPlaceAddresses;
private List[] mLikelyPlaceAttributions;
private LatLng[] mLikelyPlaceLatLngs;

private Button satelliteButton;
private Button hybridButton;
private Button mapButton;

@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);

    // Retrieve location and camera position from saved instance state.
    if (savedInstanceState != null)
    {
        mLastKnownLocation = savedInstanceState.getParcelable(KEY_LOCATION);
        mCameraPosition = savedInstanceState.getParcelable(KEY_CAMERA_POSITION);
    }

    setContentView(R.layout.activity_maps);

    // Construct a PlacesClient
    Places.initialize(getApplicationContext(), "");
    mPlacesClient = Places.createClient( context: this);

    // Construct a FusedLocationProviderClient.
    mFusedLocationProviderClient = LocationServices.getFusedLocationProviderClient( activity: this);

    // Build the map.
    SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
            .findFragmentById(R.id.map);

    mapFragment.getMapAsync( onMapReadyCallback: this);

    this.satelliteButton = findViewById(R.id.satelliteButton);
    this.hybridButton = findViewById(R.id.hybridButton);
    this.mapButton = findViewById(R.id.mapButton);
}

/**
 * Saves the state of the map when the activity is paused.
 */
@Override
protected void onSaveInstanceState(Bundle outState)
{
    if (mMap != null)
    {
        outState.putParcelable(KEY_CAMERA_POSITION, mMap.getCameraPosition());
        outState.putParcelable(KEY_LOCATION, mLastKnownLocation);
        super.onSaveInstanceState(outState);
    }
}
```

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        getMenuInflater().inflate(R.menu.current_place_menu, menu);
        return true;
    }

    /**
     * Handles a click on the menu option to get a place.
     * @param item The menu item to handle.
     * @return Boolean.
     */
    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        if (item.getItemId() == R.id.option_get_place)
        {
            showCurrentPlace();
        }
        return true;
    }

    /**
     * Manipulates the map when it's available.
     * This callback is triggered when the map is ready to be used.
     */
    @Override
    public void onMapReady(GoogleMap map)
    {
        mMap = map;

        // Use a custom info window adapter to handle multiple lines of text in the
        // info window contents.
        mMap.setInfoWindowAdapter(new GoogleMap.InfoWindowAdapter()
        {
            @Override
            // Return null here, so that getInfoContents() is called next.
            public View getInfoWindow(Marker arg0) { return null; }

            //No entiendo muy bien esto
            @Override
            public View getInfoContents(Marker marker)
            {
                // Inflate the layouts for the info window, title and snippet.
                View infoWindow = getLayoutInflater().inflate(R.layout.custom_info_contents,
                        (FrameLayout) findViewById(R.id.map), attachToRoot: false);

                TextView title = infoWindow.findViewById(R.id.title);
                title.setText(marker.getTitle());

                TextView snippet = infoWindow.findViewById(R.id.snippet);
                snippet.setText(marker.getSnippet());

                return infoWindow;
            }
        });
```

```
        // Prompt the user for permission.
        getLocationPermission();

        // Turn on the My Location layer and the related control on the map.
        updateLocationUI();

        // Get the current location of the device and set the position of the map.
        getDeviceLocation();

        //Draw Lines
        DrawLine();
    }

    /**
     * Gets the current location of the device, and positions the map's camera.
     */
    private void getDeviceLocation()
    {
        /*
         * Get the best and most recent location of the device, which may be null in rare
         * cases when a location is not available.
         */
        try
        {
            if (mLocationPermissionGranted)
            {
                //Aquí se obtiene la última localización conocida del dispositivo
                Task<Location> locationResult = mFusedLocationProviderClient.getLastLocation();

                locationResult.addOnCompleteListener( activity: this, (task) -> {
                        if (task.isSuccessful())
                        {
                            // Set the map's camera position to the current location of the device.
                            mLastKnownLocation = task.getResult();

                            if (mLastKnownLocation != null)
                            {
                                mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(
                                        new LatLng(mLastKnownLocation.getLatitude(),
                                                mLastKnownLocation.getLongitude()), DEFAULT_ZOOM));
                            }
                        }

                        else {
                            Log.d(TAG,  msg: "Current location is null. Using defaults.");
                            Log.e(TAG,  msg: "Exception: %s", task.getException());
                            mMap.moveCamera(CameraUpdateFactory
                                    .newLatLngZoom(mDefaultLocation, DEFAULT_ZOOM));
                            mMap.getUiSettings().setMyLocationButtonEnabled(false);
                        }
                    });
            }
        }

        catch (SecurityException e)  {
            Log.e( tag: "Exception: %s", e.getMessage());
        }
    }
```

```java
private void getLocationPermission()
{
    /*
     * Request location permission, so that we can get the location of the
     * device. The result of the permission request is handled by a callback,
     * onRequestPermissionsResult.
     */

    if (ContextCompat.checkSelfPermission(this.getApplicationContext(),
            android.Manifest.permission.ACCESS_FINE_LOCATION)
            == PackageManager.PERMISSION_GRANTED)
    {
        mLocationPermissionGranted = true;
    }

    else
    {
        ActivityCompat.requestPermissions( activity: this,
                new String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},
                PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION);
    }
}

/**
 * Handles the result of the request for location permissions.
 */
@Override
public void onRequestPermissionsResult(int requestCode,
                                       @NonNull String[] permissions,
                                       @NonNull int[] grantResults)
{
    mLocationPermissionGranted = false;

    switch (requestCode)
    {
        case PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION:
        {
            // If request is cancelled, the result arrays are empty.
            if (grantResults.length > 0
                    && grantResults[0] == PackageManager.PERMISSION_GRANTED)
            {
                mLocationPermissionGranted = true;
            }
        }
    }

    updateLocationUI();
}
```

```
private void showCurrentPlace()
{
    if (mMap == null)
    {
        return;
    }

    if (mLocationPermissionGranted)
    {
        // Use fields to define the data types to return.
        List<Place.Field> placeFields = Arrays.asList(Place.Field.NAME, Place.Field.ADDRESS,
                Place.Field.LAT_LNG);

        // Use the builder to create a FindCurrentPlaceRequest.
        FindCurrentPlaceRequest request =
                FindCurrentPlaceRequest.newInstance(placeFields);

        // Get the likely places - that is, the businesses and other points of interest that
        // are the best match for the device's current location.
        @SuppressWarnings("MissingPermission") final
        Task<FindCurrentPlaceResponse> placeResult =
                mPlacesClient.findCurrentPlace(request);

        placeResult.addOnCompleteListener ((task) -> {
                if (task.isSuccessful() && task.getResult() != null)
                {
                    FindCurrentPlaceResponse likelyPlaces = task.getResult();

                    // Set the count, handling cases where less than 5 entries are returned.
                    int count;

                    if (likelyPlaces.getPlaceLikelihoods().size() < M_MAX_ENTRIES) {
                        count = likelyPlaces.getPlaceLikelihoods().size();
                    } else {
                        count = M_MAX_ENTRIES;
                    }

                    int i = 0;
                    mLikelyPlaceNames = new String[count];
                    mLikelyPlaceAddresses = new String[count];
                    mLikelyPlaceAttributions = new List[count];
                    mLikelyPlaceLatLngs = new LatLng[count];

                    for (PlaceLikelihood placeLikelihood : likelyPlaces.getPlaceLikelihoods())
                    {
                        // Build a list of likely places to show the user.
                        mLikelyPlaceNames[i] = placeLikelihood.getPlace().getName();
                        mLikelyPlaceAddresses[i] = placeLikelihood.getPlace().getAddress();
                        mLikelyPlaceAttributions[i] = placeLikelihood.getPlace()
                                .getAttributions();
                        mLikelyPlaceLatLngs[i] = placeLikelihood.getPlace().getLatLng();
```

```
                    i++;
                    if (i > (count - 1)) {
                        break;
                    }
                }

                // Show a dialog offering the user the list of likely places, and add a
                // marker at the selected place.
                MapsActivity.this.openPlacesDialog();
            }
            else {
                Log.e(TAG,  msg: "Exception: %s", task.getException());
            }
        });
    }

    else {
        // The user has not granted permission.
        Log.i(TAG,  msg: "The user did not grant location permission.");

        // Add a default marker, because the user hasn't selected a place.
        mMap.addMarker(new MarkerOptions()
                .title("Default Location")
                .position(mDefaultLocation)
                .snippet("No places found, because location permission is disabled."));

        // Prompt the user for permission.
        getLocationPermission();
    }
}

/**
 * Displays a form allowing the user to select a place from a list of likely places.
 */
private void openPlacesDialog()
{
    // Ask the user to choose the place where they are now.
    DialogInterface.OnClickListener listener = (dialog, which) -> {
        // The "which" argument contains the position of the selected item.
        LatLng markerLatLng = mLikelyPlaceLatLngs[which];
        String markerSnippet = mLikelyPlaceAddresses[which];

        if (mLikelyPlaceAttributions[which] != null) {
            markerSnippet = markerSnippet + "\n" + mLikelyPlaceAttributions[which];
        }

        // Add a marker for the selected place, with an info window
        // showing information about that place.
        mMap.addMarker(new MarkerOptions()
                .title(mLikelyPlaceNames[which])
                .position(markerLatLng)
                .snippet(markerSnippet));
```

```
            // Position the map's camera at the location of the marker.
            mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(markerLatLng,
                    DEFAULT_ZOOM));
        };

        // Display the dialog.
        AlertDialog dialog = new AlertDialog.Builder( context: this)
                .setTitle("Choose a place")
                .setItems(mLikelyPlaceNames, listener)
                .show();
    }

    /**
     * Updates the map's UI settings based on whether the user has granted location permission.
     */
    private void updateLocationUI()
    {
        if (mMap == null)
            return;

        try
        {
            if (mLocationPermissionGranted)
            {
                mMap.setMyLocationEnabled(true);
                mMap.getUiSettings().setMyLocationButtonEnabled(true);
            }

            else {
                mMap.setMyLocationEnabled(false);
                mMap.getUiSettings().setMyLocationButtonEnabled(false);
                mLastKnownLocation = null;
                getLocationPermission();
            }
        }

        catch (SecurityException e)
        {
            Log.e( tag: "Exception: %s", e.getMessage());
        }
    }

    public void DrawLine()
    {
        PolylineOptions polylineOptions = new PolylineOptions();
        polylineOptions.add(new LatLng( v: 37.423143,  v1: -122.083868))
                .add(new LatLng( v: 37.428405,  v1: -122.077294))
                .add(new LatLng( v: 37.424704,  v1: -122.041000));
        mMap.addPolyline(polylineOptions);
    }
```

```
    //Método para poner la vista en modo satélite
    public void setMapMode(View view) { mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL); }

    //Método para poner la vista en modo satélite
    public void setSatelliteMode(View view) { mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE); }

    //Método para poner la vista en modo satélite
    public void setHybridMode(View view) { mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID); }
}
```

Por último, destacar que toda esta información ha sido obtenida de la documentación de Google Maps SDK.

5.1.3. Integración y uso de Firebase en Android y Node JS

Para la realización de este tutorial necesitaremos tener instalado en nuestro equipo Android Studio y Node JS. Para el caso de Android Studio también será necesario instalar las siguientes herramientas adicionales al SDK de Android, tal y como se indicó en el primer tutorial de este Apéndice A: "Google Play Services", "Android SDK Tools", "NDK (Side by side)".

Además de los dos programas anteriores, debemos disponer de un editor de texto para programación, como puede ser Visual Studio Code, que nos vaya a facilitar la implementación de nuestro servidor en Node JS.

Una vez tengamos todo nuestro equipo totalmente configurado, ya podremos iniciar los siguientes pasos del tutorial.

En primer lugar, crearemos un nuevo proyecto en Android Studio y, según la documentación de Firebase, necesitaremos que la aplicación pueda funcionar con dispositivos Android con una versión de la API número 16. Además, se recomienda la creación de la "MainActivity" como de tipo 'Empty'.

Cuando dispongamos de nuestro proyecto de Android Studio ya creado, debemos dirigirnos al fichero "Android Manifest.xml", para añadir el siguiente permiso:

<uses-permission android:name="android.permission.INTERNET" />

Este permiso será fundamental para realizar el envío y recepción de notificaciones a través de Firebase, por lo que necesitaremos conexión a Internet.

Además, debemos declarar un servicio dentro de nuestro objeto 'application', para definir una clase que funcionará como un servicio, es decir, será una clase que se encontrará en ejecución mientras la aplicación se encuentre en funcionamiento. La función principal de esta clase será escuchar y atender todas las notificaciones, que lleguen a la aplicación desde los servidores de Firebase. Por lo tanto, debemos declarar el servicio de la siguiente manera:

```
<!-- [START firebase_service] -->
<service
    android:name=".firebase.MyFirebaseMessagingService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
<!-- [END firebase_service] -->
```

Cuando tengamos completado el paso anterior tan sólo tendremos que agregar el siguiente código como primer hijo del elemento 'application'. A través del código que se muestra a continuación, podremos definir el color y el icono que utilizaremos para mostrarle al usuario una nueva notificación:

```
<!-- Set custom default icon. This is used when no icon is set for incoming
notification messages. -->
<meta-data
    android:name="com.google.firebase.messaging.default_notification_icon"
    android:resource="@drawable/ic_notifications_active_white_24dp" />

<!-- Set color used with incoming notification messages. This is used when no
color is set for the incoming
     notification message. -->
<meta-data
    android:name="com.google.firebase.messaging.default_notification_color"
    android:resource="@color/buttonColor" />
```

Por último, debemos modificar los dos ficheros "build.gradle", es decir tanto el 'build.gradle (Project: nombreProyecto)' como el 'build.gradle (Module:app)'. En el primero de ellos, dentro del campo dependencias añadiremos lo siguiente:

classpath 'com.google.gms:google-services:4.2.0'

Mediante el uso de la dependencia anterior podremos agregar a nuestro proyecto distintos servicios, que Google nos ofrece para utilizar en Android de una manera sencilla, entre los que podemos encontrar Firebase.

Una vez modificado este fichero debemos dirigirnos al 'build.gradle (Module:app)'. En dicho fichero añadiremos las siguientes dependencias dentro del campo destinado para ello:

implementation 'com.google.firebase:firebase-analytics:17.4.1'
implementation 'com.google.firebase:firebase-messaging:20.1.7'

Finalmente, sincronizaremos el proyecto para que Gradle pueda descargar todos aquellos paquetes que podamos necesitar durante el desarrollo de nuestra aplicación.
El siguiente paso será, la creación del servicio que previamente habremos declarado en nuestro fichero 'Manifest'. Después agregaremos el siguiente código para mostrar las notificaciones enviadas mediante Firebase:

```java
public class MyFirebaseMessagingService extends FirebaseMessagingService
{
    private static final String TAG = "MyFirebaseMessService";
    public static final String SERVICE_MESSAGE = "Service Message";
    public static final String MISSION_START = "FirebaseMessagingService Start Mission";
    public static final String MISSION_KEY = "order";


    @Override
    public void onNewToken(@Nullable String token)
    {
        super.onNewToken(token);
        Log.d(TAG,  msg: "onNewToken: " + token);
    }

    @Override
    public void onMessageReceived(@Nullable RemoteMessage remoteMessage)
    {
        super.onMessageReceived(remoteMessage);
        Log.d(TAG,  msg: "Mensaje Recibido");

        String title = "";
        String message = "";

        Map<String, String> data = remoteMessage.getData();

        if(data.size() > 0)
        {
            title = data.get("title");
            message = data.get("message");
        }

        else
        {
            RemoteMessage.Notification notification = remoteMessage.getNotification();
            title = notification.getTitle();
            message = notification.getBody();
        }

        sendNotification(title, message);

        //Enviamos la señal de iniciar la misión
        sendStartMissionOrder(title);
    }
```

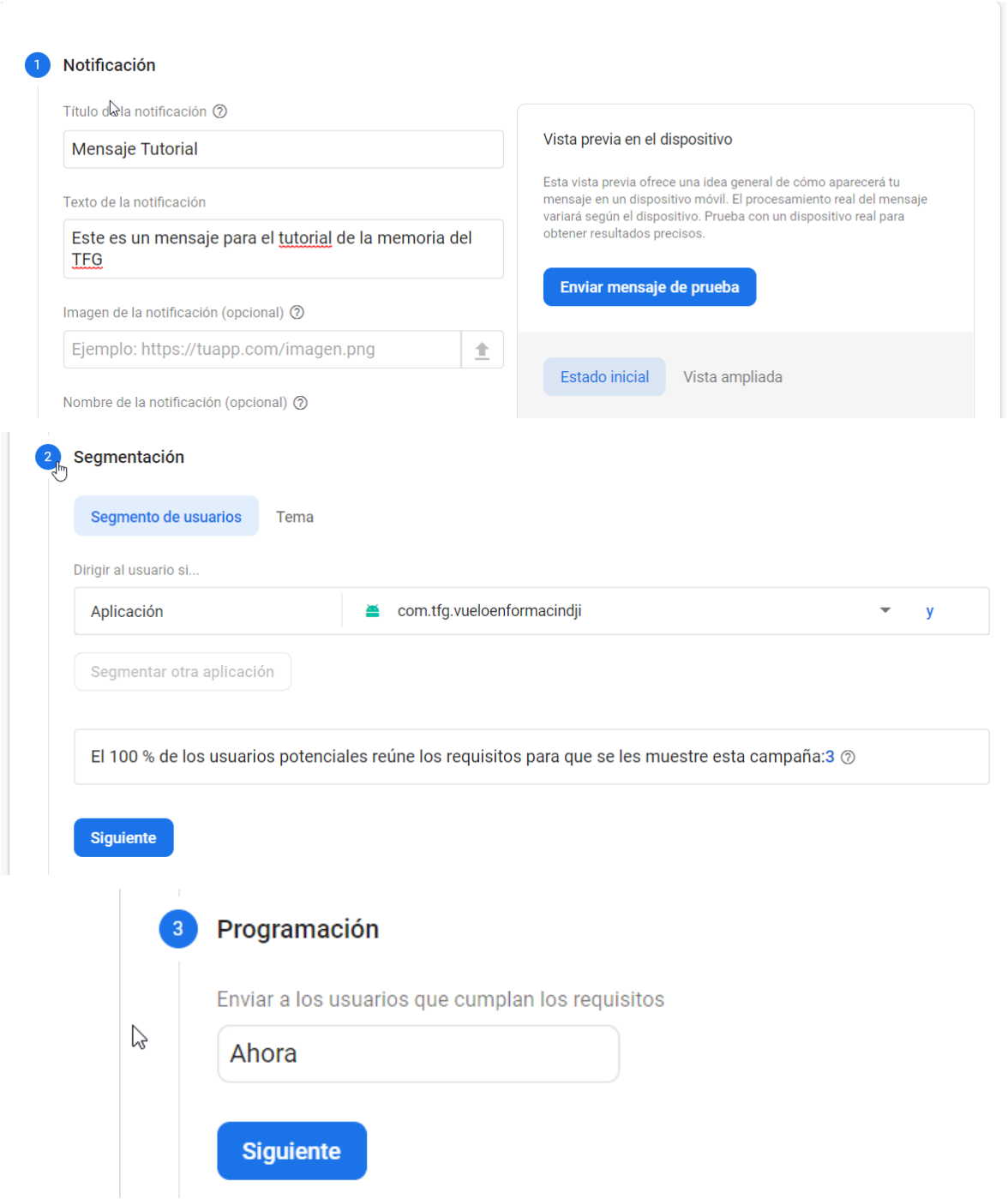
```
    private void sendNotification(String title, String message)
    {
        Intent intent = new Intent( packageContext: this, MissionActivity.class);
        PendingIntent pendingIntent = PendingIntent.getActivity( context: this,
                MyNotification.NOTIFICATION_ID, intent, PendingIntent.FLAG_ONE_SHOT);

        MyNotification notification = new MyNotification( context: this,
                MyNotification.CHANNEL_ID_NOTIFICATIONS);
        notification.build(R.drawable.ic_launcher_foreground, title, message, pendingIntent);
        notification.addChannel( chanelName: "Notificaciones", NotificationManager.IMPORTANCE_DEFAULT);
        notification.show(MyNotification.NOTIFICATION_ID);
    }

    //Método para mandar un mensaje al MissionActivity
    private void sendStartMissionOrder(String title)
    {
        Intent intent = new Intent(SERVICE_MESSAGE);
        intent.putExtra(MISSION_KEY, title);

        LocalBroadcastManager.getInstance(getApplicationContext())
                .sendBroadcast(intent);
    }

}
```

Una vez completado el procedimiento anterior, ya deberíamos disponer de una aplicación a la que podemos enviar notificaciones desde Firebase, para ello previamente habremos creado un nuevo proyecto en Firebase con la finalidad de poder realizar comprobaciones.

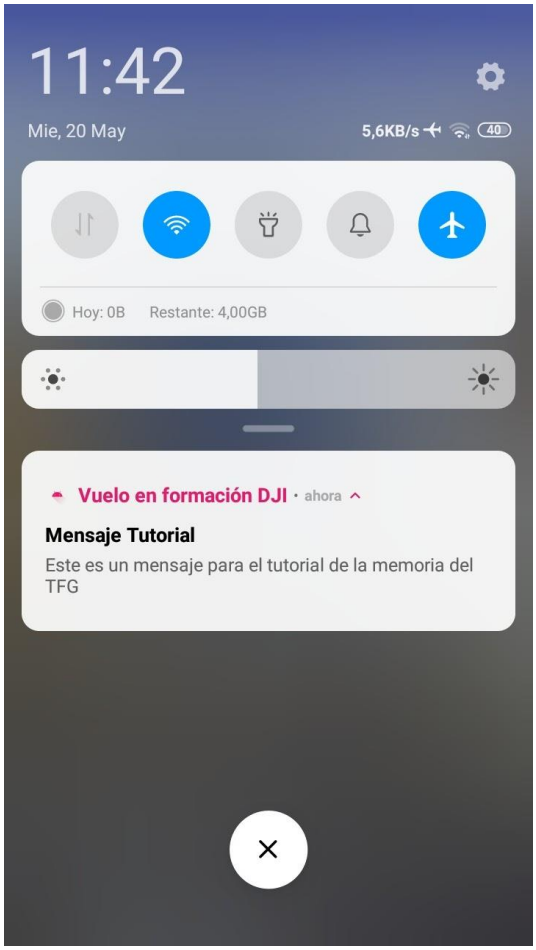
Para crear un nuevo proyecto en Firebase, debemos dirigirnos al aparato "Ir a Consola" que tiene una opción para generar un proyecto o añadir la aplicación a uno existente. Seguidamente, debemos añadir a nuestro proyecto la aplicación que acabamos de crear. Para ello, rellenaremos todos los campos del formulario de acuerdo con el nombre de la aplicación y al nombre del paquete principal de la misma. Una vez registrada nuestra aplicación, podremos dirigirnos al apartado de 'CloudMessaging'. Dicho apartado lo encontramos sobre el menú lateral izquierdo en la sección de Crecimiento.

A continuación, podremos generar un nuevo mensaje desde Firebase y comprobar si, dicho mensaje llega a nuestro terminal móvil.

El mensaje desde la consola de Firebase lo podemos generar de la siguiente manera:



Si todo el proceso que se ha explicado previamente ha funcionado correctamente deberíamos obtener un resultado como el que se muestra a continuación:



El siguiente paso consistirá en la configuración de nuestro servidor en Node JS, para realizar el envío de mensajes utilizando Firebase como elemento intermediario. Por ello, y tal y como se indica en la documentación de Firebase, será necesario disponer de una versión de Node JS superior al número 10. En el caso de disponer de una versión de Node JS válida para nuestro propósito podemos utilizar el siguiente comando para crear el proyecto:

npm init

Una vez generado nuestro proyecto, debemos instalar “express” en una versión superior a la 4.12. Finalmente, instalaremos Firebase en nuestro proyecto de Node JS, mediante el siguiente comando:

npm install firebase-admin --save

Cuando termine de instalarse el paquete anterior, ya dispondremos de nuestro proyecto totalmente configurado para comenzar a implementar el envío de notificaciones mediante Firebase. Por ello, en nuestro fichero principal del proyecto, añadiremos el módulo de Firebase.

var admin = require('firebase-admin');

Además del fichero principal del servidor, procederemos a crear uno nuevo en el que definiremos todo lo necesario para realizar el envío, es decir, tendremos un fichero principal donde podremos implementar todas las funciones que nuestro servidor vaya a necesitar y, en un otro adicional implementaremos todo el proceso necesario para realizar el envío de las notificaciones a los terminales Android.

En cuanto a la implementación del fichero auxiliar podemos utilizar la que se muestra a continuación:

```
var admin = require("firebase-admin");
var serviceAccount = require("/home/angel/Documentos/PROYECTO-TFG/firebase.json");

admin.initializeApp({
    credential: admin.credential.cert(serviceAccount),
    databaseURL: "https://trabajo-final-de-grado-dc50b.firebaseio.com"
});

function sendPushToMission(notification)
{
    const message = {
        data: {
            title: notification.title,
            message: notification.message,
            mission: notification.mission
        },
        tokens: notification.tokens,
    }

    sendMessage(message);
}

module.exports = { sendPushToOneUser, sendPushToMission }

function sendMessage(message)
{
    admin.messaging().sendMulticast(message)
        .then((response) =>
        {
            console.log("Successfully sent message: ", response);
        })
        .catch((error) =>
        {
            console.log("Error sending message: ", error);
        });
}
```

Por último, en cuanto al procedimiento a utilizar para enviar los mensajes desde el fichero principal del servidor, simplemente debemos generar un objeto JSON con todos los parámetros del mensaje, para pasar dicho mensaje a la función encargada de su envío, tal y como se puede apreciar en la siguiente imagen:

```
const data = {
    mission: req.body.idMission,
    title: "Inicio Misión",
    message: "Inicio Misión para los drones de la misión: " + req.body.idMission,
    tokens: tokensMission
}

notification.sendPushToMission(data);
```

Por último, destacar que toda esta información ha sido obtenida de la documentación de Firebase para Android, así como de la documentación de Firebase para Node JS.

5.2. Apéndice B. Implementación Cliente

Connection Activity (Parte Lógica):

```
public class ConnectionActivity extends Activity implements View.OnClickListener
{
    /**
     * Create the layout UI elements variables
     */
    private TextView connectionStatusView;
    private TextView productInformatioView;
    private Button openButton;

    /**
     * Attributes needed for the class
     */
    private static final String TAG = ConnectionActivity.class.getName();
    private static final String[] REQUIRED_PERMISSION_LIST = new String[]{
            Manifest.permission.VIBRATE,
            Manifest.permission.INTERNET,
            Manifest.permission.ACCESS_WIFI_STATE,
            Manifest.permission.WAKE_LOCK,
            Manifest.permission.ACCESS_COARSE_LOCATION,
            Manifest.permission.ACCESS_NETWORK_STATE,
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.CHANGE_WIFI_STATE,
            Manifest.permission.WRITE_EXTERNAL_STORAGE,
            Manifest.permission.BLUETOOTH,
            Manifest.permission.BLUETOOTH_ADMIN,
            Manifest.permission.READ_EXTERNAL_STORAGE,
            Manifest.permission.READ_PHONE_STATE,
    };
    private List<String> missingPermission = new ArrayList<>();
    private AtomicBoolean isRegistrationInProgress = new AtomicBoolean( initialValue: false);
    private static final int REQUEST_PERMISSION_CODE = 12345;

    /**
     * Métodos ON
     */

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        //Primero deberemos de comprobar los permisos
        checkAndRequestPermissions();

        setContentView(R.layout.activity_connection);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
```

```
        //Después, procedemos a inicializar los objetos de la parte gráfica
        initUI();

        // Register the broadcast receiver for receiving the device connection's changes.
        IntentFilter filter = new IntentFilter();
        filter.addAction(DJI_TFG_Application.FLAG_CONNECTION_CHANGE);
        registerReceiver(mReceiver, filter);

        /**
         * Esto es lo que carga un activity por defecto
         */
        Intent intent = new Intent( packageContext: this, MissionActivity.class);
        startActivity(intent);
    }

    @Override
    public void onResume()
    {
        Log.e(TAG,  msg: "onResume");
        super.onResume();
    }

    @Override
    public void onPause()
    {
        Log.e(TAG,  msg: "onPause");
        super.onPause();
    }

    @Override
    public void onStop()
    {
        Log.e(TAG,  msg: "onStop");
        super.onStop();
    }

    public void onReturn(View view)
    {
        Log.e(TAG,  msg: "onReturn");
        this.finish();
    }

    @Override
    protected void onDestroy()
    {
        Log.e(TAG,  msg: "onDestroy");
        unregisterReceiver(mReceiver);
        super.onDestroy();
    }
```

```java
/**
 * Override the onClick() method to implement the Button's click action.
 */
@Override
public void onClick(View view)
{
    switch (view.getId())
    {
        case R.id.openButton: {
            int status = GoogleApiAvailability.getInstance().
                    isGooglePlayServicesAvailable( context: this);

            if(status != ConnectionResult.SUCCESS)
            {
                GooglePlayServicesUtil.getErrorDialog(status,  activity: this, status);
                showToast( toastMsg: "Cannot run without Google Play, please check! ");
            }

            else {
                Intent intent = new Intent( packageContext: this, MissionActivity.class);
                startActivity(intent);

                /*Intent auxIntent = new Intent(this, CreateMission.class);
                auxIntent.putExtra("djiAircraft", getProductInformatioView());
                startActivity(auxIntent);*/
            }
            break;
        }
    }
}



/**
 * Métodos de inicialización y de permisos
 */

/**
 * Implement the initUI() method to initialize the two TextViews and the Button.
 *
 * Then invoke setOnClickListener() method of button and pass this as the param.
 */
private void initUI()
{
    this.connectionStatusView = findViewById(R.id.connectionStatusView);
    this.productInformatioView = findViewById(R.id.productInformatioView);
    this.openButton = findViewById(R.id.openButton);

    this.openButton.setOnClickListener(this);
    this.openButton.setEnabled(false);
}
```

```java
/**
 * Result of runtime permission request
 */
@Override
public void onRequestPermissionsResult(int requestCode,
                                       @NonNull String[] permissions,
                                       @NonNull int[] grantResults)
{
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);

    // Check for granted permission and remove from missing list
    if (requestCode == REQUEST_PERMISSION_CODE)
    {
        for (int i = grantResults.length - 1; i >= 0; i--)
        {
            if (grantResults[i] == PackageManager.PERMISSION_GRANTED)
                missingPermission.remove(permissions[i]);
        }
    }
    // If there is enough permission, we will start the registration
    if (missingPermission.isEmpty())
        startSDKRegistration();

    else
        showToast( toastMsg: "Missing permissions!!!");
}

/**
 * Checks if there is any missing permissions, and
 * requests runtime permission if needed.
 */
private void checkAndRequestPermissions()
{
    // Check for permissions
    for (String eachPermission : REQUIRED_PERMISSION_LIST)
    {
        if (ContextCompat.checkSelfPermission( context: this, eachPermission)
                != PackageManager.PERMISSION_GRANTED)
            missingPermission.add(eachPermission);
    }

    // Request for missing permissions
    if (!missingPermission.isEmpty() && Build.VERSION.SDK_INT >= Build.VERSION_CODES.M)
    {
        ActivityCompat.requestPermissions( activity: this,
                missingPermission.toArray(new String[missingPermission.size()]),
                REQUEST_PERMISSION_CODE);
    }
}
```

```
/**
 * Method to register the application.
 */
private void startSDKRegistration()
{
    //Si el proceso de registro no se ha llevado acabo entra
    if (isRegistrationInProgress.compareAndSet( expect: false,  update: true))
    {
        AsyncTask.execute(new Runnable()
        {
            @Override
            public void run() {
                showToast( toastMsg: "registering, pls wait...");
                DJISDKManager.getInstance().registerApp(getApplicationContext(),
                        new DJISDKManager.SDKManagerCallback()
                {
                    /*
                        If the registration is successful, invoke the startConnectionToProduct()
                        method of DJISDKManager inside the onRegister() callback method to start
                        the connection between SDK and the DJI Products.
                     */
                    @Override
                    public void onRegister(DJIError djiError)
                    {
                        if (djiError == DJISDKError.REGISTRATION_SUCCESS)
                        {
                            DJILog.e( tag: "App registration",
                                    DJISDKError.REGISTRATION_SUCCESS.getDescription());
                            DJISDKManager.getInstance().startConnectionToProduct();
                            showToast( toastMsg: "Register Success");
                        }
                        else
                            showToast( toastMsg: "Register sdk fails, check network " +
                                    "is available");

                        Log.v(TAG, djiError.getDescription());
                    }

                    @Override
                    public void onProductDisconnect()
                    {
                        Log.d(TAG,  msg: "onProductDisconnect");
                        showToast( toastMsg: "Product Disconnected");
                    }

                    @Override
                    public void onProductConnect(BaseProduct baseProduct)
                    {
                        Log.d(TAG, String.format("onProductConnect newProduct:%s", baseProduct));
                        showToast( toastMsg: "Product Connected");
                    }
```

```
                    @Override
                    public void onComponentChange(BaseProduct.ComponentKey componentKey,
                                                  BaseComponent oldComponent,
                                                  BaseComponent newComponent)
                    {

                        if (newComponent != null)
                        {
                            newComponent.setComponentListener(
                                    new BaseComponent.ComponentListener()
                            {
                                @Override
                                public void onConnectivityChange(boolean isConnected)
                                {
                                    Log.d(TAG,  msg: "onComponentConnectivityChanged: "
                                            + isConnected);
                                }
                            });
                        }
                        Log.d(TAG,
                                String.format("onComponentChange key:%s, oldComponent:%s, " +
                                                "newComponent:%s",
                                        componentKey,
                                        oldComponent,
                                        newComponent));
                    }

                    @Override
                    public void onInitProcess(DJISDKInitEvent djisdkInitEvent, int i)
                    {

                    }

                    @Override
                    public void onDatabaseDownloadProgress(long l, long l1)
                    {

                    }
                });
            }
        });
    }
}

/**
 * We check the BaseProduct's connection status by invoking isConnected() method.
 * If the product is connected, we enable the button, update the
 * status text content and update the product content with product name.
 * Otherwise, if the product is disconnected, we disable the mBtnOpen button and update
 * the mTextProduct and mTextConnectionStatus textViews' content.
 */
```

```
private void refreshSDKRelativeUI()
{
    BaseProduct mProduct = DJI_TFG_Application.getProductInstance();

    if (null != mProduct && mProduct.isConnected())
    {
        Log.v(TAG,  msg: "refreshSDK: True");
        this.openButton.setEnabled(true);

        String str = mProduct instanceof Aircraft ? "DJIAircraft" : "DJIHandHeld";
        String aux = "Status: " + str + " connected";
        this.connectionStatusView.setText(aux);

        if (null != mProduct.getModel())
        {
            aux = "" + mProduct.getModel().getDisplayName();
            this.productInformatioView.setText(aux);
        }

        else
            this.productInformatioView.setText("Product Information");
    }

    else {

        Log.v(TAG,  msg: "refreshSDK: False");
        this.openButton.setEnabled(false);

        this.productInformatioView.setText("Product Information");
        this.connectionStatusView.setText("Status: No Product Connected");
    }
}
```

Connection Activity (Parte Gráfica):

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/foto_activity_1_2"
    android:orientation="vertical">

    <TextView
        android:id="@+id/applicationNameView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Vuelo en formación DJI"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textSize="20sp"
        android:textColor="@android:color/black"
        android:gravity="center"
        android:layout_centerHorizontal="true"
        android:textStyle="bold"
        android:layout_marginTop="70dp"/>

    <TextView
        android:id="@+id/connectionStatusView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="Status: Product Disconnected"
        android:textColor="@android:color/black"
        android:textSize="20sp"
        android:textStyle="normal"
        android:layout_alignTop="@+id/applicationNameView"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="100dp" />
```

```
    <TextView
        android:id="@+id/productInformatioView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:text="DJI Aircraft"
        android:textColor="@android:color/black"
        android:textSize="20sp"
        android:gravity="center"
        android:textStyle="normal"
        android:layout_alignBottom="@+id/openButton"
        android:layout_marginBottom="120dp"/>

    <Button
        android:id="@+id/openButton"
        android:layout_width="120dp"
        android:layout_height="55dp"
        android:layout_centerHorizontal="true"
        android:background="@drawable/round_btn"
        android:text="Open"
        android:textColor="@android:color/white"
        android:textSize="20sp"
        android:layout_alignBottom="@+id/myNameView"
        android:layout_marginBottom="90dp"/>

    <TextView
        android:id="@+id/myNameView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:gravity="center"
        android:text="Ángel Pérez Arroyo"
        android:textColor="?android:attr/colorForeground"
        android:textSize="14sp"
        android:textStyle="normal"
        android:layout_marginBottom="30dp"
        android:layout_alignParentBottom="true"/>
</RelativeLayout>
```

Create Mission Activity (Parte Lógica):

```
public class CreateMission extends Activity implements View.OnClickListener
{
    /**
     * Create the layout UI elements variables
     */
    private TextView urlText;
    private TextView idMission;
    private Button accessButton;
    private Button createMissionButton;
    private TextView errorServerText;
    private String djiAircraft;


    /**
     * Attributes needed for the class
     */
    protected static final String TAG = "MainActivity";
    protected static final String HTTP = "http://";

    /**
     * Métodos ON
     */
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.create_mission);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);

        this.djiAircraft = getIntent().getStringExtra( name: "djiAircraft");

        //Inicializamos los elementos de la parte gráfica
        initUI();
    }

    @Override
    public void onResume()
    {
        Log.e(TAG,  msg: "onResume");
        super.onResume();
    }

    @Override
    public void onPause()
    {
        Log.e(TAG,  msg: "onPause");
        super.onPause();
    }
```

```
@Override
public void onStop()
{
    Log.e(TAG, msg: "onStop");
    super.onStop();
}

public void onReturn(View view)
{
    Log.e(TAG, msg: "onReturn");
    this.finish();
}

@Override
protected void onDestroy()
{
    Log.e(TAG, msg: "onDestroy");
    super.onDestroy();
}

@Override
public void onClick(View v)
{
    switch (v.getId())
    {
        case R.id.accessButton:
        {
            checkIDmission();
            break;
        }

        case R.id.createMissionButton:
        {
            createNewMission();
            break;
        }
    }
}



/**
 * Métodos de inicialización y de permisos
 */

/**
 * This method allow us to init the 7 Button variables and implement their
 * setOnClickListener method and pass "this" as parameter.
 */
private void initUI()
{
    this.urlText = findViewById(R.id.urlText);
    this.idMission = findViewById(R.id.idMissionNumber);
    this.accessButton = findViewById(R.id.accessButton);
```

```
        this.createMissionButton = findViewById(R.id.createMissionButton);
        this.errorServerText = findViewById(R.id.errorServerText);

        //Obtenemos las preferencias del usuario
        SharedPreferences preferences = getSharedPreferences( name: "preferencias",
                Context.MODE_PRIVATE);
        String aux = preferences.getString( key: "url",  defValue: "");

        if(!aux.equals(""))
            setUrlString(aux);

        this.accessButton.setOnClickListener(this);
        this.createMissionButton.setOnClickListener(this);
    }

    //Método para guardar las preferencias
    public void savePreferencies()
    {
        String auxText = getUrlString();

        //Esto es para editar el fichero 'preferencias'
        SharedPreferences preferences = getSharedPreferences( name: "preferencias",
                Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = preferences.edit();
        editor.putString("url", auxText);
        editor.apply();
    }

    //Método para comprobar si el id de misión introducido por el usuario existe en el servidor
    private void checkIDmission()
    {
        if(!getUrlString().equals("") && !getIDmission().equals(""))
        {
            // Instantiate the RequestQueue.
            RequestQueue queue = Volley.newRequestQueue( context: this);
            String url = HTTP + getUrlString() + "/checkMissionID/" + getIDmission();

            // Request a string response from the provided URL.
            StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
                    new Response.Listener<String>()
                    {
                        @Override
                        public void onResponse(String response)
                        {
                            runOnUiThread(new Runnable()
                            {
                                @Override
                                public void run()
                                {
                                    //Como ha ido bien el registro paso a guardar la url del
                                    // servidor introducida por el usuario
                                    savePreferencies();
```

```
                                Intent intent = new Intent(getApplicationContext(),
                                        MainActivity.class);
                                intent.putExtra( name: "url", getUrlString());
                                intent.putExtra( name: "idMission", getIDmission());
                                intent.putExtra( name: "djiAircraft", getDjiAircraft());
                                startActivity(intent);
                            }
                        });
                    }

                }, new Response.ErrorListener()
        {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                runOnUiThread(new Runnable()
                {
                    @Override
                    public void run()
                    {
                        closeKeyboard();
                        setErrorServerText("La misión " + getIDmission() +
                                " no existe o no se puede" +
                                "contactar con el servidor");
                        errorServerText.setVisibility(View.VISIBLE);
                    }
                });

                Log.d(TAG,  msg: "Respuesta: " + error.toString());
            }
        });

        // Add the request to the RequestQueue.
        queue.add(stringRequest);
    }

    else
        showToast( toastMsg: "Se deben rellenar los dos campos anteriores");
}

//Método para crear una nueva misión
private void createNewMission()
{
    try
    {
        RequestQueue requestQueue = Volley.newRequestQueue( context: this);
        String URL = HTTP + getUrlString() + "/createMission";

        JSONObject jsonBody = new JSONObject();
        final String requestBody = jsonBody.toString();
```

```
        StringRequest stringRequest = new StringRequest(Request.Method.POST, URL,
                new Response.Listener<String>() {
            @Override
            public void onResponse(String response)
            {
                //Coordenadas recibidas correctamente por el servidor
                if(response.equals("200"))
                {
                    //Como ha ido bien el registro paso a guardar la url del servidor
                    // introducida por el usuario
                    savePreferencies();
                    getLastMission();
                }

                Log.i( tag: "OnResponse newMission", response);
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                //Saco el mensaje de error
                showToast( toastMsg: "Hay un problema de conexión con el servidor");
                Log.e( tag: "onErrorResponse", error.toString());
            }
        }) {
            @Override
            public String getBodyContentType() { return "application/json; charset=utf-8"; }

            @Override
            public byte[] getBody() throws AuthFailureError {
                return requestBody == null ? null : requestBody.getBytes(StandardCharsets.UTF_8)
            }

            @Override
            protected Response<String> parseNetworkResponse(NetworkResponse response) {
                String responseString = "";
                if (response != null) {
                    responseString = String.valueOf(response.statusCode);
                    // can get more details such as response.headers
                }
                return Response.success(responseString, HttpHeaderParser.
                        parseCacheHeaders(response));
            }
        };

        requestQueue.add(stringRequest);

    }

    catch (Exception e)
    {
        showToast( toastMsg: "Error al crear una nueva misión");
        Log.d(TAG,  msg: "Salta excepción al crear una nueva misión");
    }
}
```

```
//Método para obtener la última misión creada
private void getLastMission()
{
    try
    {
        // Instantiate the RequestQueue.
        RequestQueue queue = Volley.newRequestQueue( context: this);
        String url = HTTP + getUrlString() + "/getLastMission";

        // Request a string response from the provided URL.
        StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
                new Response.Listener<String>()
        {
            @Override
            public void onResponse(String response)
            {
                try
                {
                    JSONObject jsonObject = new JSONObject(response);
                    JSONArray jsonArray = jsonObject.getJSONArray( name: "result");
                    String auxMissionID = jsonArray.getJSONObject( index: 0).
                            getString( name: "max(`idMission`)");

                    Intent intent = new Intent(getApplicationContext(), MainActivity.class);
                    intent.putExtra( name: "url", getUrlString());
                    intent.putExtra( name: "idMission", auxMissionID);
                    intent.putExtra( name: "djiAircraft", getDjiAircraft());
                    startActivity(intent);
                }

                catch (JSONException e) {
                    e.printStackTrace();
                }
            }

        }, new Response.ErrorListener()
        {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                //textView.setText("That didn't work!");
                Log.d(TAG,  msg: "onErrorResponse de getServerWaypoints: " + error.toString());
            }
        });

        // Add the request to the RequestQueue.
        queue.add(stringRequest);
    }

    catch (Exception e) {

        showToast( toastMsg: "Error al obtener la última misión");
        Log.d(TAG,  msg: "Salta excepción al obtener la última misión");
    }
}
```

```
    /**
     * Métodos Auxiliares
     */
    //Getters
    public TextView getUrlText() { return urlText; }

    public String getUrlString() { return getUrlText().getText().toString(); }

    public String getIDmission() { return idMission.getText().toString(); }

    //Setters
    public void setUrlString(String urlText) { this.urlText.setText(urlText); }

    public void setErrorServerText(String errorServerText) {
        this.errorServerText.setText(errorServerText);
    }

    public void closeKeyboard()
    {
        View auxView = this.getCurrentFocus();

        if(auxView != null)
        {
            InputMethodManager inputMethodManager = (InputMethodManager)getSystemService(
                    Context.INPUT_METHOD_SERVICE);
            inputMethodManager.hideSoftInputFromWindow(auxView.getWindowToken(),  flags: 0);
        }
    }

    public String getDjiAircraft() { return djiAircraft; }

    private void showToast(final String toastMsg)
    {
        runOnUiThread(new Runnable()
        {
            @Override
            public void run()
            {
                Toast.makeText(getApplicationContext(), toastMsg, Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

Create Mission Activity (Parte Gráfica):

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/military_jet"
    android:fillViewport="true"
    android:scrollbars="none"
    android:orientation="vertical">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <TextView
            android:id="@+id/applicationNameView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Vuelo en formación DJI"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textSize="22sp"
            android:textColor="@android:color/white"
            android:gravity="center"
            android:layout_centerHorizontal="true"
            android:textStyle="normal"
            android:layout_marginTop="70dp" />

        <EditText
            android:id="@+id/urlText"
            android:layout_width="250dp"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:background="@drawable/rounded_corner"
            android:hint="URL del Servidor"
            android:textColorHint="@android:color/white"
            android:textColor="@android:color/white"
            android:gravity="center"
            android:textSize="18sp"
            android:paddingTop="5sp"
            android:paddingBottom="5sp"
            android:layout_alignBottom="@+id/applicationNameView"
            android:layout_marginBottom="-100dp"/>
```

```
    <EditText
        android:id="@+id/idMissionNumber"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:background="@drawable/rounded_corner"
        android:gravity="center"
        android:hint="Insert your ID Mission"
        android:inputType="number"
        android:paddingTop="5sp"
        android:paddingBottom="5sp"
        android:textColor="@android:color/white"
        android:textColorHint="@android:color/white"
        android:textSize="18sp"
        android:layout_alignBottom="@+id/urlText"
        android:layout_marginBottom="-100dp"/>

    <Button
        android:id="@+id/accessButton"
        android:layout_width="wrap_content"
        android:layout_height="50dp"
        android:layout_centerHorizontal="true"
        android:background="@drawable/round_btn"
        android:text="Acceder"
        android:textColor="@android:color/white"
        android:textSize="18sp"
        android:paddingStart="5dp"
        android:paddingEnd="5dp"
        android:layout_alignBottom="@+id/idMissionNumber"
        android:layout_marginBottom="-80dp"/>

    <TextView
        android:id="@+id/errorServerText"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/accessButton"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="-70dp"
        android:gravity="center"
        android:textColor="@android:color/holo_red_light"
        android:textSize="20sp"
        android:textStyle="normal"
        android:visibility="invisible" />
```

```
    <Button
        android:id="@+id/createMissionButton"
        android:layout_width="wrap_content"
        android:layout_height="50dp"
        android:layout_centerHorizontal="true"
        android:background="@drawable/round_btn"
        android:text="Crear Nueva Misión"
        android:textColor="@android:color/white"
        android:textSize="18sp"
        android:paddingStart="5dp"
        android:paddingEnd="5dp"
        android:layout_alignBottom="@+id/errorServerText"
        android:layout_marginBottom="-140dp"/>

    <TextView
        android:id="@+id/myNameView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_alignBottom="@+id/createMissionButton"
        android:layout_marginBottom="-80dp"
        android:gravity="center"
        android:text="Ángel Pérez Arroyo"
        android:textColor="@color/black"
        android:textSize="14sp"
        android:textStyle="normal" />

</RelativeLayout>
```

MainActivity (Parte Lógica):

```
public class MainActivity extends Activity implements View.OnClickListener
{
    /**
     * Create the layout UI elements variables
     */

    private Spinner userType;
    private Button registerButton;
    private TextView errorRegisterView;


    /**
     * Attributes needed for the class
     */
    protected static final String TAG = "MainActivity";
    protected static final String HTTP = "http://";
    private String[] userTypes = {"Tipo de Usuario", "Usuario", "Administrador"};
    protected static String URL = null;
    protected static String idMission = null;
    private String djiAircraft;
    private String idDrone;

    /**
     * Métodos ON
     */
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);

        URL = getIntent().getStringExtra( name: "url");
        idMission = getIntent().getStringExtra( name: "idMission");
        djiAircraft = getIntent().getStringExtra( name: "djiAircraft");

        //Inicializamos los elementos de la parte gráfica
        initUI();
    }

    @Override
    public void onResume()
    {
        Log.e(TAG,  msg: "onResume");
        super.onResume();
    }
```

```
@Override
public void onPause()
{
    Log.e(TAG, msg: "onPause");
    super.onPause();
}

@Override
public void onStop()
{
    Log.e(TAG, msg: "onStop");
    super.onStop();
}

public void onReturn(View view)
{
    Log.e(TAG, msg: "onReturn");
    this.finish();
}

@Override
protected void onDestroy()
{
    Log.e(TAG, msg: "onDestroy");
    super.onDestroy();
}

@Override
public void onClick(View view)
{
    switch (view.getId())
    {
        case R.id.registerButton:
        {
            if(checkFields())
            {
                //Registramos a un usuario
                registerDrone();
            }

            else
                showToast( toastMsg: "Se debe indicar la URL del servidor y el tipo de usuario");
        }
    }
}
```

```
    /**
     * Métodos de inicialización y de permisos
     */

    /**
     * This method allow us to init the 7 Button variables and implement their
     * setOnClickListener method and pass "this" as parameter.
     */
    private void initUI()
    {
        this.userType = findViewById(R.id.userType);
        this.registerButton = findViewById(R.id.registerButton);
        this.errorRegisterView = findViewById(R.id.errorRegisterView);

        //Inicializamos el Spinner
        initSpinner();

        this.registerButton.setOnClickListener(this);
    }

    //Método para hacer inicializar el spinner
    private void initSpinner()
    {...}




    /**
     * Métodos Auxiliares
     */

    //Getter
    public Spinner getUserType() { return userType; }

    public String getUserTypeString() { return this.userType.getSelectedItem().toString(); }

    public TextView getDronesRegisterView() { return errorRegisterView; }

    public String getDronesRegisterText() { return getDronesRegisterView().getText().toString(); }

    public static String getURL() { return URL; }
    public static String getIdMission() { return idMission; }

    public String getDjiAircraft() { return djiAircraft; }

    public String getIdDrone() { return idDrone; }
```

```
    //Setters
    public void setDronesRegisterView(TextView errorRegisterView) {
        this.errorRegisterView = errorRegisterView;
    }

    public void setDronesRegisterString(String errorRegisterText) {
        this.errorRegisterView.setText(errorRegisterText);
    }

    public static void setURL(String URL) { MainActivity.URL = URL; }

    public void setIdDrone(String idDrone) { this.idDrone = idDrone; }



    //Método para comprobar que los campos se han rellenado de manera correcta antes de intentar
    // registrar
    private boolean checkFields()
    {
        String url = getURL();
        String userType = getUserTypeString();

        //Si ambos campos se han rellenado puedo proceder
        if(!url.equals("") && !userType.equals("Tipo de Usuario"))
            return true;

        else
            return false;
    }

    //Método para registrar a un usuario
    private void registerDrone()
    {
        try
        {
            RequestQueue requestQueue = Volley.newRequestQueue( context: this);
            String URL = HTTP + getURL() + "/register";

            JSONObject jsonBody = new JSONObject();
            jsonBody.put( name: "userType", getUserTypeString());
            jsonBody.put( name: "idMission", getIdMission());
            jsonBody.put( name: "djiAircraft", getDjiAircraft());
            final String requestBody = jsonBody.toString();

            StringRequest stringRequest = new StringRequest(Request.Method.POST, URL,
                    new Response.Listener<String>() {
                @Override
                public void onResponse(String response)
                {
                    //No hay un usuario administrador registrado
                    if(response.equals("200"))
                    {
```

```
                runOnUiThread(new Runnable()
                {
                    @Override
                    public void run()
                    {
                        //Saco el mensaje de error
                        setDronesRegisterString("Se debe registrar en primer lugar un " +
                                "usuario administrador");
                        errorRegisterView.setVisibility(View.VISIBLE);
                    }

                });
            }

            //Drone registrado correctamente
            else if(response.equals("201"))
                getDroneID();

            //Caso en el que no se admiten más drones
            else if(response.equals("202"))
            {
                runOnUiThread(new Runnable()
                {
                    @Override
                    public void run()
                    {
                        //Saco el mensaje de error
                        setDronesRegisterString("Pide al administrador que amplie el " +
                                "número de drones admitidos");
                        errorRegisterView.setVisibility(View.VISIBLE);
                    }

                });
            }

            //Caso en el que se intenta registrar un segundo administrador
            else if(response.equals("203"))
            {
                runOnUiThread(new Runnable()
                {
                    @Override
                    public void run()
                    {
                        //Saco el mensaje de error
                        setDronesRegisterString("Sólo se puede registrar a un " +
                                "administrador por misión");
                        errorRegisterView.setVisibility(View.VISIBLE);
                    }

                });
            }

            Log.i( tag: "OnResponse", response);
        }
```

```
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                //Saco el mensaje de error
                setDronesRegisterString("Hay un problema de conexión con el servidor");
                errorRegisterView.setVisibility(View.VISIBLE);
                Log.e( tag: "onErrorResponse", error.toString());
            }
        }) {
            @Override
            public String getBodyContentType() { return "application/json; charset=utf-8"; }

            @Override
            public byte[] getBody() throws AuthFailureError {
                return requestBody == null ? null : requestBody.getBytes(StandardCharsets.UTF_8);
            }

            @Override
            protected Response<String> parseNetworkResponse(NetworkResponse response) {
                String responseString = "";
                if (response != null) {
                    responseString = String.valueOf(response.statusCode);
                    // can get more details such as response.headers
                }
                return Response.success(responseString, HttpHeaderParser.
                        parseCacheHeaders(response));
            }
        };

        requestQueue.add(stringRequest);
    }

    catch (Exception e)
    {
        setDronesRegisterString("Error al registrar el drone.");
        this.errorRegisterView.setVisibility(View.VISIBLE);
        Log.d(TAG,  msg: "Salta excepción en el registro del drone");
    }
}

//Método para obtener el id del drone
private void getDroneID()
{
    // Instantiate the RequestQueue.
    RequestQueue queue = Volley.newRequestQueue( context: this);
    String url = HTTP + getURL() + "/getDroneID/" + getUserTypeString() + "/" +
            getDjiAircraft() + "/" + getIdMission();

    // Request a string response from the provided URL.
    StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>()
            {
```

```
                @Override
                public void onResponse(String response)
                {
                    try
                    {
                        JSONObject jsonObject = new JSONObject(response);
                        JSONArray jsonArray = jsonObject.getJSONArray( name: "result");
                        String idDroneResponse = jsonArray.getJSONObject( index: 0).
                                getString( name: "max(`idDrone`)");
                        setIdDrone(idDroneResponse);

                        runOnUiThread(new Runnable()
                        {
                            @Override
                            public void run()
                            {
                                //Después cambio y cierro la activity
                                Intent intent;

                                if(getUserTypeString().equals("Administrador"))
                                    intent = new Intent(getApplicationContext(),
                                            FlyRegisterActivity.class);

                                else
                                    intent = new Intent(getApplicationContext(),
                                            MissionActivity.class);

                                intent.putExtra( name: "userType", getUserTypeString());
                                intent.putExtra( name: "url", getURL());
                                intent.putExtra( name: "idMission", getIdMission());
                                intent.putExtra( name: "djiAircraft", getDjiAircraft());
                                intent.putExtra( name: "idDrone", getIdDrone());
                                startActivity(intent);
                                finish();
                            }

                        });
                    }

                    catch (JSONException e) {
                        e.printStackTrace();
                    }
                }

            }, new Response.ErrorListener()
        {
```

```
            @Override
            public void onErrorResponse(VolleyError error)
            {
                runOnUiThread(new Runnable()
                {
                    @Override
                    public void run()
                    {
                        showToast( toastMsg: "La misión " + getIdMission() +
                                " no existe o no se puede" +
                                "contactar con el servidor");
                    }
                });

                Log.d(TAG,  msg: "Respuesta: " + error.toString());
            }
        });

        // Add the request to the RequestQueue.
        queue.add(stringRequest);
    }

    private void showToast(final String toastMsg)
    {
        runOnUiThread(new Runnable()
        {
            @Override
            public void run()
            {
                Toast.makeText(getApplicationContext(), toastMsg, Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

MainActivity (Parte Gráfica):

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/foto_activity_2"
    android:orientation="vertical">

    <TextView
        android:id="@+id/applicationNameView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Vuelo en formación DJI"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textSize="22sp"
        android:textColor="@android:color/black"
        android:gravity="center"
        android:layout_centerHorizontal="true"
        android:textStyle="normal"
        android:layout_marginTop="50dp"/>

    <Spinner
        android:id="@+id/userType"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:focusable="auto"
        android:hint="Tipo de Usuario"
        android:textColor="@android:color/white"
        android:gravity="center"
        android:layout_alignTop="@+id/applicationNameView"
        android:layout_marginTop="100dp"
        android:popupBackground="@null"/>

    <Button
        android:id="@+id/registerButton"
        android:layout_width="120dp"
        android:layout_height="55dp"
        android:layout_centerHorizontal="true"
        android:background="@drawable/round_btn"
```

```
        android:text="Registrar"
        android:textColor="@android:color/white"
        android:textSize="18sp"
        android:layout_alignBottom="@+id/errorRegisterView"
        android:layout_marginBottom="80dp"/>

    <TextView
        android:id="@+id/errorRegisterView"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:gravity="center"
        android:text="Drones registrados:"
        android:textColor="@android:color/holo_red_light"
        android:textSize="18sp"
        android:textStyle="normal"
        android:visibility="invisible"
        android:layout_marginBottom="50dp"
        android:layout_alignBottom="@+id/myNameView"/>


    <TextView
        android:id="@+id/myNameView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="30dp"
        android:gravity="center"
        android:text="Ángel Pérez Arroyo"
        android:textColor="#4259D6"
        android:textSize="14sp"
        android:textStyle="normal" />

</RelativeLayout>
```

FlyRegisterActivity (Parte Lógica):

```
public class FlyRegisterActivity extends Activity implements View.OnClickListener
{
    /**
     * Create the layout UI elements variables
     */
    private static Spinner dronesNumber;
    private Spinner dronesDistance;
    private Spinner minumunAltitude;
    private Spinner altitudeBetweenDrones;
    private Button backButton;
    private Button updateButton;
    private Button removeButton;

    /**
     * Attributes needed for the class
     */
    protected static final String TAG = "FlyRegisterActivity";
    protected static final String HTTP = "http://";
    protected static String URL = null;
    protected static String userType;
    protected static final String[] maxDroneNumber = {"Número de Drones","1","2","3","4","5","6",
            "7","8","9","10"};
    protected static final String[] distance = {"Distancia entre drones (m)","1","2","3","4","5"};
    protected static final String[] minAltitud = {"Altura mínima de vuelo (m)",
            "20","30","40","50","60","70","80","90","95","100"};
    protected static final String[] altitud = {"Altura entre drones (m)","1","2","3","4","5"};
    protected static String idMission = null;
    private static String djiAircraft;
    private String idDrone;

    /**
     * Métodos ON
     */
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fly_register);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);

        //Obtengo la url introducida en la activity anterior
        URL = getIntent().getStringExtra( name: "url");
        userType = getIntent().getStringExtra( name: "userType");
        idMission = getIntent().getStringExtra( name: "idMission");
        djiAircraft = getIntent().getStringExtra( name: "djiAircraft");
        idDrone = getIntent().getStringExtra( name: "idDrone");

        //Inicializamos los elementos de la parte gráfica
        initUI();
    }
```

```
    /**
     * Métodos de inicialización y de permisos
     */

    /**
     * This method allow us to init the 7 Button variables and implement their
     * setOnClickListener method and pass "this" as parameter.
     */
    private void initUI() {
        this.dronesNumber = findViewById(R.id.dronesNumber);
        this.dronesDistance = findViewById(R.id.dronesDistance);
        this.minumunAltitude = findViewById(R.id.minumunAltitude);
        this.altitudeBetweenDrones = findViewById(R.id.altitudeBetweenDrones);
        this.backButton = findViewById(R.id.backButton);
        this.updateButton = findViewById(R.id.updateButton);
        this.removeButton = findViewById(R.id.removeButton);

        //Inicializamos los Spinners
        initSpinnerDrones();
        initSpinnerDistance();
        initSpinnerMinAltutud();
        initSpinnerAltitud();

        this.backButton.setOnClickListener(this);
        this.updateButton.setOnClickListener(this);
        this.removeButton.setOnClickListener(this);
    }

    /**
     * Métodos Auxiliares
     */

    //Getters
    public static String getDronesNumber() { return dronesNumber.getSelectedItem().toString(); }

    public String getDronesDistance() { return dronesDistance.getSelectedItem().toString(); }

    public String getMinumunAltitude() { return minumunAltitude.getSelectedItem().toString(); }

    public String getAltitudeBetweenDrones() {
        return altitudeBetweenDrones.getSelectedItem().toString();
    }

    public static String getURL() { return URL; }

    public static String getUserType() { return userType; }

    public static String getIdMission() { return idMission; }

    public String getDjiAircraft() { return djiAircraft; }

    public String getIdDrone() { return idDrone; }
```

```
//Método para comprobar que todos los campos se rellenan
private boolean checkSpinnerValues()
{

    if(!getDronesNumber().equals("Número de Drones") &&
            !getDronesDistance().equals("Distancia entre drones (m)") &&
            !getMinumunAltitude().equals("Altura mínima de vuelo (m)") &&
            !getAltitudeBetweenDrones().equals("Altura entre drones (m)"))
        return true;

    else
        return false;
}

//Método para actualizar los datos de la misión
private void updateMissionData()
{
    try
    {
        RequestQueue requestQueue = Volley.newRequestQueue( context: this);
        String URL = HTTP + getURL() + "/updateMissionData";

        JSONObject jsonBody = new JSONObject();
        jsonBody.put( name: "dronesNumber", getDronesNumber());
        jsonBody.put( name: "dronesDistance", getDronesDistance());
        jsonBody.put( name: "minumunAltitude", getMinumunAltitude());
        jsonBody.put( name: "altitudeBetweenDrones", getAltitudeBetweenDrones());
        jsonBody.put( name: "idMission", getIdMission());
        final String requestBody = jsonBody.toString();

        StringRequest stringRequest = new StringRequest(Request.Method.PUT, URL,
                new Response.Listener<String>() {
            @Override
            public void onResponse(String response)
            {
                //No hay un usuario administrador registrado
                if(response.equals("200"))
                {
                    runOnUiThread(new Runnable()
                    {
                        @Override
                        public void run()
                        {
                            showToast( toastMsg: "Misión actualizada correctamente");

                            Intent intent = new Intent(getApplicationContext(),
                                    MissionActivity.class);
                            intent.putExtra( name: "userType", getUserType());
                            intent.putExtra( name: "url", getURL());
                            intent.putExtra( name: "idMission", getIdMission());
                            intent.putExtra( name: "djiAircraft", getDjiAircraft());
                            intent.putExtra( name: "idDrone", getIdDrone());
                            startActivity(intent);
                        }
```

```
                    });
                }

                Log.i( tag: "OnResponse", response);
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                //Saco el mensaje de error
                showToast( toastMsg: "Hay un problema de conexión con el servidor");
                Log.e( tag: "onErrorResponse", error.toString());
            }
        }) {
            @Override
            public String getBodyContentType() { return "application/json; charset=utf-8"; }

            @Override
            public byte[] getBody() throws AuthFailureError {
                return requestBody == null ? null : requestBody.getBytes(StandardCharsets.UTF_8);
            }

            @Override
            protected Response<String> parseNetworkResponse(NetworkResponse response) {
                String responseString = "";
                if (response != null) {
                    responseString = String.valueOf(response.statusCode);
                    // can get more details such as response.headers
                }
                return Response.success(responseString, HttpHeaderParser.
                        parseCacheHeaders(response));
            }
        };

        requestQueue.add(stringRequest);
    }

    catch (Exception e)
    {
        showToast( toastMsg: "Error al actualizar los datos de la misión");
        Log.d(TAG,  msg: "Salta excepción al actualizar los datos de la misión");
    }
}

/**
 * Spinners
 */
//Método para hacer inicializar el spinner
private void initSpinnerDrones()
{...}

//Método para hacer inicializar de la distancia entre drones
private void initSpinnerDistance()
{...}
```

```
    //Método para hacer inicializar el spinner de la altura mínima de vuelo
    private void initSpinnerMinAltutud()
    {...}

    //Método para hacer inicializar el spinner de la altura entre drones
    private void initSpinnerAltitud()
    {...}


    private void showToast(final String toastMsg)
    {
        runOnUiThread(new Runnable()
        {
            @Override
            public void run()
            {
                Toast.makeText(getApplicationContext(), toastMsg, Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

FlyRegisterActivity (Parte Gráfica):

```
<?xml version="1.0" encoding="utf-8"?>

<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/ScrollView01"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fillViewport="true"
    android:scrollbars="none"
    android:background="@drawable/phantom_4">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <TextView
            android:id="@+id/applicationNameView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Vuelo en formación DJI"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:textSize="22sp"
            android:textColor="@android:color/black"
            android:gravity="center"
            android:layout_centerHorizontal="true"
            android:textStyle="normal"
            android:layout_marginTop="40dp"/>

        <!-- Poner el spinner de 1-10 -->
        <Spinner
            android:id="@+id/dronesNumber"
            android:layout_width="250dp"
            android:layout_height="wrap_content"
            android:layout_alignTop="@+id/applicationNameView"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="60dp"
            android:focusable="auto"
            android:gravity="center"
            android:textColor="@android:color/white"
            android:textSize="16sp"
            android:popupBackground="@null"/>
```

```
    <Spinner
        android:id="@+id/dronesDistance"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:gravity="center"
        android:layout_marginTop="60dp"
        android:textSize="16sp"
        android:layout_alignTop="@+id/dronesNumber"
        android:popupBackground="@null"/>

    <Spinner
        android:id="@+id/minumunAltitude"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:gravity="center"
        android:layout_marginTop="60dp"
        android:textSize="16sp"
        android:layout_alignTop="@+id/dronesDistance"
        android:popupBackground="@null"/>

    <Spinner
        android:id="@+id/altitudeBetweenDrones"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:gravity="center"
        android:layout_marginTop="60dp"
        android:textSize="16sp"
        android:layout_alignTop="@+id/minumunAltitude"
        android:popupBackground="@null"/>

    <com.tfg.vueloenformacindji.auxiliary_classes.droneDrawView
        android:id="@+id/customView"
        android:layout_width="match_parent"
        android:layout_height="260dp"
        android:layout_marginBottom="70dp"
        android:layout_marginTop="50dp"
        android:layout_marginLeft="15dp"
        android:layout_marginRight="15dp"
        android:background="#7B575252"
        android:layout_alignTop="@+id/altitudeBetweenDrones"/>
```

```
    <LinearLayout
        android:id="@+id/linearButtons"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/customView"
        android:layout_marginBottom="-80dp"
        android:layout_centerHorizontal="true"
        android:gravity="center">

        <Button
            android:id="@+id/backButton"
            android:layout_width="wrap_content"
            android:layout_height="50dp"
            android:background="@drawable/round_btn"
            android:drawableBottom="@drawable/ic_arrow_back_white_32dp"
            android:paddingBottom="9dp" />

        <Button
            android:id="@+id/updateButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="15dp"
            android:layout_marginEnd="15dp"
            android:background="@drawable/round_btn"
            android:text="Actualizar"
            android:textColor="@android:color/white"
            android:textSize="20sp" />

        <Button
            android:id="@+id/removeButton"
            android:layout_width="wrap_content"
            android:layout_height="50dp"
            android:background="@drawable/round_btn"
            android:drawableBottom="@drawable/ic_clear_white_32dp"
            android:paddingBottom="9dp" />
    </LinearLayout>

    <LinearLayout
        android:id="@+id/linearMyName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/linearButtons"
        android:layout_alignParentEnd="true"
        android:layout_marginTop="100dp"
        android:gravity="center">
```

```
            <TextView
                android:id="@+id/myNameView"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:text="Ángel Pérez Arroyo"
                android:textColor="@android:color/white"
                android:textSize="14sp"
                android:textStyle="normal" />
        </LinearLayout>

    </RelativeLayout>
</ScrollView>
```

MissionActivity (Parte Lógica):

```
public class MissionActivity extends AppCompatActivity implements OnMapReadyCallback,
        View.OnClickListener, GoogleMap.OnMapClickListener
{
    /**
     * Create the layout UI elements variables
     */
    private Button backButton;
    private Button missionButton;
    private Button droneCameraButton;
    private Button waypointButton;
    private Button mapViewButton;
    private Button addWaypointsButton;
    private Button clearWaypointsButton;
    private TextView userID;
    Spinner spinnerSpeed;

    /**
     * Attributes needed for the class
     */
    protected static final String[] velocidad = {"Velocidad de misión","Velocidad Baja",
            "Velocidad Media","Velocidad Alta"};
    private static final String TAG = MissionActivity.class.getSimpleName();
    private GoogleMap mMap;
    private CameraPosition mCameraPosition;

    private static final int DEFAULT_ZOOM = 17;
    private static final int PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION = 1;
    private boolean mLocationPermissionGranted;

    // The geographical location where the device is currently located. That is, the last-known
    // location retrieved by the Fused Location Provider.
    private Location mLastKnownLocation;

    // The entry point to the Fused Location Provider.
    private FusedLocationProviderClient mFusedLocationProviderClient;

    private final LatLng mDefaultLocation = new LatLng( v: 38.220305, v1: -0.516853);
    private final Map<Integer, Marker> mMarkers = new ConcurrentHashMap<~>();

    //Variables para las misiones
    private float altitude = 30.0f;
    private float mSpeed = 10.0f;
    public static WaypointMission.Builder waypointMissionBuilder;
    private FlightController mFlightController;
    private WaypointMissionOperator instance;

    private WaypointMissionFinishedAction mFinishedAction = WaypointMissionFinishedAction.GO_HOME;
    private WaypointMissionHeadingMode mHeadingMode = WaypointMissionHeadingMode.AUTO;
    private double droneLocationLat = 40.220305, droneLocationLng = -0.926853;
    private Marker droneMarker = null;
    private List<Waypoint> waypointList = new ArrayList<>();
```

```
    private ArrayList<Pair<Double, Double>> sendWaypoints = new ArrayList<~>();
    private ArrayList<Pair<Double, Double>> receivedWaypoints = new ArrayList<~>();

    // Keys for storing activity state.
    private static final String KEY_CAMERA_POSITION = "camera_position";
    private static final String KEY_LOCATION = "location";

    protected static String URL = null;
    protected static final String HTTP = "http://";
    protected static String userType;
    protected static String idMission = null;
    private static String djiAircraft;
    private String idDrone;

    //Variables Auxiliares
    private Boolean auxMissionSpeed = false;
    private Boolean auxFlyAltitud = false;

    private String firebaseToken = "";

    /**
     * Métodos ON
     */
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);

        //Obtengo la url introducida en la activity anterior
        URL = "192.168.100.50:9090";
        userType = "Usuario";
        idMission = "2";
        djiAircraft = "Mavic Air";
        idDrone = "1";

        IntentFilter filter = new IntentFilter();
        filter.addAction(DJI_TFG_Application.FLAG_CONNECTION_CHANGE);
        registerReceiver(mReceiver, filter);

        // Retrieve location and camera position from saved instance state.
        if (savedInstanceState != null)
        {
            mLastKnownLocation = savedInstanceState.getParcelable(KEY_LOCATION);
            mCameraPosition = savedInstanceState.getParcelable(KEY_CAMERA_POSITION);
        }

        setContentView(R.layout.activity_mission);

        // Construct a FusedLocationProviderClient.
        mFusedLocationProviderClient = LocationServices.getFusedLocationProviderClient( activity: this);

        // Build the map.
        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
                .findFragmentById(R.id.map);

        mapFragment.getMapAsync( onMapReadyCallback: this);
```

```
        //Inicializamos los componentes de la parte gráfica
        initUI();

        //Obtengo el número de drones registrados hasta el momento
        getDronesRegistered();

        //Añado Listener a los waypoints
        addListener();

        //Para registrarse en un tema
        getFirebaseInstance();
        //registerFirebaseTopic();
    }

    @Override
    public void onStart()
    {
        super.onStart();

        LocalBroadcastManager.getInstance(getApplicationContext())
                .registerReceiver(mMissionReceiver, new IntentFilter(MyFirebaseMessagingService.
                        SERVICE_MESSAGE));
    }

    /**
     * Manipulates the map when it's available.
     * This callback is triggered when the map is ready to be used.
     */
    @Override
    public void onMapReady(GoogleMap map)
    {
        if (mMap == null)
        {
            mMap = map;
            setUpMap();
        }

        // Prompt the user for permission.
        getLocationPermission();

        // Turn on the My Location layer and the related control on the map.
        updateLocationUI();

        // Get the current location of the device and set the position of the map.
        getDeviceLocation();

        //Cargo la vista satélite por defecto
        setSatelliteMode();
    }
```

```java
    /**
     * The onMapClick() method will be invoked when user tap on the Map View.
     *
     * When user tap on different position of the Map View, we will create a MarkerOptions object
     * and assign the "LatLng" object to it, then invoke "gMap"'s addMarker() method by passing
     * the markerOptions parameter to add the waypoint markers on the Google map.
     */
    @Override
    public void onMapClick(LatLng point)
    {
        if(getUserType().equals("Administrador"))
        {
            if (getAddWaypointsText().equals("Exit"))
                markWaypoint(point);

            else
                showToast( toastMsg: "Cannot Add Waypoint");
        }
    }

    /**
     * Saves the state of the map when the activity is paused.
     */
    @Override
    protected void onSaveInstanceState(Bundle outState)
    {
        if (mMap != null)
        {
            outState.putParcelable(KEY_CAMERA_POSITION, mMap.getCameraPosition());
            outState.putParcelable(KEY_LOCATION, mLastKnownLocation);
            super.onSaveInstanceState(outState);
        }
    }

    @Override
    public void onClick(View v)
    {
        switch (v.getId())
        {
            case R.id.backButton:
            {
                finish();
                break;
            }

            case R.id.missionButton:
            {
                //Abro el popup
                openPopupMission();
                break;
            }
```

```
        case R.id.droneCameraButton:
        {
            Intent uxView = new Intent( packageContext: this, UXactivity.class);
            startActivity(uxView);
            break;
        }

        case R.id.addWaypointsButton:
        {
            enableDisableAdd();
            break;
        }

        case R.id.clearWaypointsButton:
        {
            this.mMap.clear();
            this.sendWaypoints.clear();
            this.receivedWaypoints.clear();

            waypointList.clear();
            //waypointMissionBuilder.waypointList(waypointList);
            updateDroneLocation();
            break;
        }

        case R.id.waypointsButton:
        {
            getServerWaypoints();

            break;
        }

        case R.id.mapType:
        {
            getMapChargeView();
            break;
        }

        //Botones del layout de la misión
        case R.id.sendWaypointsButton:
        {
            sendWaypoints();
            break;
        }

        case R.id.geoButton:
        {
            Intent intent = new Intent(getApplicationContext(), GEO_Activity.class);
            startActivity(intent);

            break;
        }
```

```
            case R.id.startButton:
            {
                sendStartMission();
                break;
            }

            case R.id.stopButton:
            {
                sendStopMission();
                break;
            }

            case R.id.pauseButton:
            {
                sendResumePauseMission();
                break;
            }

            //Botones del layout de seleccionar el mapa
            case R.id.mapButton:
            {
                setMapMode();
                break;
            }

            case R.id.satelliteButton:
            {
                setSatelliteMode();
                break;
            }

            case R.id.reliefButton:
            {
                setReliefMode();
                break;
            }

        }
    }


    @Override
    protected void onResume()
    {
        super.onResume();
        initFlightController();
    }

    @Override
    protected void onPause()
    {
        super.onPause();

        LocalBroadcastManager.getInstance(getApplicationContext())
                .unregisterReceiver(mMissionReceiver);
    }
```

```
@Override
protected void onDestroy()
{
    unregisterReceiver(mReceiver);
    removeListener();
    FirebaseMessaging.getInstance().unsubscribeFromTopic(getIdMission());
    super.onDestroy();
}


/**
 * Métodos de inicialización y de permisos
 */
private void initUI()
{
    this.backButton = findViewById(R.id.backButton);
    this.droneCameraButton = findViewById(R.id.droneCameraButton);
    this.waypointButton = findViewById(R.id.waypointsButton);
    this.mapViewButton = findViewById(R.id.mapType);

    this.userID = findViewById(R.id.userIDview);
    this.backButton.setOnClickListener(this);
    this.droneCameraButton.setOnClickListener(this);
    this.waypointButton.setOnClickListener(this);
    this.mapViewButton.setOnClickListener(this);


    //Pongo el botón de añadir waypoints si el usuario es administrador
    if(getUserType().equals("Administrador"))
    {
        this.missionButton = findViewById(R.id.missionButton);
        this.addWaypointsButton = findViewById(R.id.addWaypointsButton);
        this.clearWaypointsButton = findViewById(R.id.clearWaypointsButton);

        this.addWaypointsButton.setOnClickListener(this);
        this.clearWaypointsButton.setOnClickListener(this);
        this.missionButton.setOnClickListener(this);

        this.addWaypointsButton.setVisibility(View.VISIBLE);
        this.missionButton.setVisibility(View.VISIBLE);
        this.backButton.setVisibility(View.VISIBLE);
    }

    String auxID = "ID de usuario: " + getUserType();
    this.userID.setText(auxID);
}

//Método para habilitar las funciones de administrador
private void enableDisableAdd()
{
    if (getAddWaypointsText().equals("Añadir Waypoints"))
    {
        this.addWaypointsButton.setText("Exit");
        this.clearWaypointsButton.setVisibility(View.VISIBLE);
    }
```

```
        else
        {
            this.addWaypointsButton.setText("Añadir Waypoints");
            this.clearWaypointsButton.setVisibility(View.GONE);
        }

    }

    /**
     * Since we need to detect the product connection status, we should register a BroadcastReceiver
     * in the onCreate() method and override the onReceive()
     */
    protected BroadcastReceiver mReceiver = (context, intent) -> {
            onProductConnectionChange();
    };

    protected BroadcastReceiver mMissionReceiver = (context, intent) -> {
            String message = intent.getStringExtra(MyFirebaseMessagingService.MISSION_KEY);

            if(message.equals("Inicio Misión"))
            {
                if(receivedWaypoints.isEmpty())
                    showToast( toastMsg: "Debes obtener primero los waypoints");

                //Le paso al drone los puntos que tiene que hacer
                else
                    setWaypointsMission();

                if(receivedWaypoints.isEmpty() || waypointList.isEmpty())
                    showToast( toastMsg: "No hay ningún waypoint añadido");

                else
                {
                    try
                    {
                        //Actualizamos la posición del drone
                        updateDroneLocation();

                        //Configuramos la misión
                        configWayPointMission();

                        //Actualizamos los parámetros de los waypoints
                        uploadWayPointMission();

                        //Ejecutamos la misión
                        startWaypointMission();
                    }

                    catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }

            else if(message.equals("Detener Misión"))
                stopWaypointMission();
```

```
        else if(message.equals("Reanudar/Pausar Misión"))
        {
            if(getWaypointMissionOperator().getCurrentState() ==
                    WaypointMissionState.EXECUTION_PAUSED)
                resumeWaypointMission();

            else if(getWaypointMissionOperator().getCurrentState() ==
                    WaypointMissionState.EXECUTING)
                pauseWaypointMission();
        }
    };

    private void onProductConnectionChange() { initFlightController(); }


    /**
     * Prompts the user for permission to use the device location.
     */
    private void getLocationPermission()
    {
        /*
         * Request location permission, so that we can get the location of the
         * device. The result of the permission request is handled by a callback,
         * onRequestPermissionsResult.
         */

        if (ContextCompat.checkSelfPermission(this.getApplicationContext(),
                android.Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED)
        {
            mLocationPermissionGranted = true;
        }

        else
        {
            ActivityCompat.requestPermissions( activity: this,
                    new String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},
                    PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION);
        }
    }

    /**
     * Handles the result of the request for location permissions.
     */
    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           @NonNull String[] permissions,
                                           @NonNull int[] grantResults)
    {
        mLocationPermissionGranted = false;

        switch (requestCode)
        {
            case PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION:
            {
```

```
                // If request is cancelled, the result arrays are empty.
                if (grantResults.length > 0
                        && grantResults[0] == PackageManager.PERMISSION_GRANTED)
                {
                    mLocationPermissionGranted = true;
                }
            }
        }

        updateLocationUI();
    }


    /**
     * Métodos Auxiliares
     */

    public static String getURL() { return URL; }

    public static String getUserType() { return userType; }

    public String getAddWaypointsText() { return addWaypointsButton.getText().toString(); }

    // Add the listener for click for amap object
    private void setUpMap() { this.mMap.setOnMapClickListener(this); }

    public ArrayList<Pair<Double, Double>> getSendWaypoints() { return sendWaypoints; }

    public ArrayList<Pair<Double, Double>> getReceivedWaypoints() { return receivedWaypoints; }

    public String getSpinnerSpeed() { return spinnerSpeed.toString(); }

    public String getSpinnerSpeedSelected() { return spinnerSpeed.getSelectedItem().toString(); }

    private void resetSendWaypoints() { this.sendWaypoints.clear(); }

    private void resetMmap() { this.mMap.clear(); }

    public static String getIdMission() { return idMission; }

    public String getDjiAircraft() { return djiAircraft; }

    public String getIdDrone() { return idDrone; }

    public static String[] getVelocidad() { return velocidad; }

    public float getAltitude() { return altitude; }

    public float getmSpeed() { return mSpeed; }

    public Boolean getAuxMissionSpeed() { return auxMissionSpeed; }

    public Boolean getAuxFlyAltitud() { return auxFlyAltitud; }


    public String getFirebaseToken() { return firebaseToken; }
```

```
    public void setAltitude(float altitude) { this.altitude = altitude; }

    public void setmSpeed(float mSpeed) { this.mSpeed = mSpeed; }

    public void setAuxMissionSpeed(Boolean auxMissionSpeed) {
        this.auxMissionSpeed = auxMissionSpeed;
    }

    public void setAuxFlyAltitud(Boolean auxFlyAltitud) { this.auxFlyAltitud = auxFlyAltitud; }

    public void setFirebaseToken(String firebaseToken) { this.firebaseToken = firebaseToken; }


    private void markWaypoint(LatLng point)
    {
        //Create MarkerOptions object
        MarkerOptions markerOptions = new MarkerOptions();
        markerOptions.position(point);
        markerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_RED));

        Marker marker = this.mMap.addMarker(markerOptions);
        mMarkers.put(mMarkers.size(), marker);

        Pair auxPair = new Pair(point.latitude, point.longitude);

        //Por último me guardo la coordenada GPS del marcador
        this.sendWaypoints.add(auxPair);
    }

    /**
     * Updates the map's UI settings based on whether the user has granted location permission.
     */
    private void updateLocationUI()
    {
        if (mMap == null)
            return;

        try
        {
            if (mLocationPermissionGranted)
            {
                mMap.setMyLocationEnabled(true);
                mMap.getUiSettings().setMyLocationButtonEnabled(true);
            }

            else {
                mMap.setMyLocationEnabled(false);
                mMap.getUiSettings().setMyLocationButtonEnabled(false);
                mLastKnownLocation = null;
                getLocationPermission();
            }
        }
```

```java
/**
 * Gets the current location of the device, and positions the map's camera.
 */
private void getDeviceLocation()
{
    /*
     * Get the best and most recent location of the device, which may be null in rare
     * cases when a location is not available.
     */
    try
    {
        if (mLocationPermissionGranted)
        {
            //Aquí se obtiene la última localización conocida del dispositivo
            Task<Location> locationResult = mFusedLocationProviderClient.getLastLocation();

            locationResult.addOnCompleteListener( activity: this, new OnCompleteListener<Location>()
            {
                @Override
                public void onComplete(@NonNull Task<Location> task)
                {
                    if (task.isSuccessful())
                    {
                        // Set the map's camera position to the current location of the device.
                        mLastKnownLocation = task.getResult();

                        if (mLastKnownLocation != null)
                        {
                            mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(
                                    new LatLng(mLastKnownLocation.getLatitude(),
                                            mLastKnownLocation.getLongitude()), DEFAULT_ZOOM));
                        }
                    }

                    else {
                        Log.d(TAG,  msg: "Current location is null. Using defaults.");
                        Log.e(TAG,  msg: "Exception: %s", task.getException());
                        mMap.moveCamera(CameraUpdateFactory
                                .newLatLngZoom(mDefaultLocation, DEFAULT_ZOOM));
                        mMap.getUiSettings().setMyLocationButtonEnabled(false);
                    }
                }
            });
        }
    }

    catch (SecurityException e)  {
        Log.e( tag: "Exception: %s", e.getMessage());
    }
}
```

```
    //Método para lanzar la vista de los distintos tipos de mapas
    private void getMapChargeView()
    {
        try
        {
            RelativeLayout relativeLayout = (RelativeLayout) getLayoutInflater().inflate(
                    R.layout.map_type_layout,  root: null);
            Button mapView = relativeLayout.findViewById(R.id.mapButton);
            Button satelliteView = relativeLayout.findViewById(R.id.satelliteButton);
            Button reliefView = relativeLayout.findViewById(R.id.reliefButton);

            mapView.setOnClickListener(this);
            satelliteView.setOnClickListener(this);
            reliefView.setOnClickListener(this);

            AlertDialog.Builder alertDialog = new AlertDialog.Builder( context: this);
            alertDialog.setTitle("");
            alertDialog.setView(relativeLayout);
            alertDialog.create().show();

        }

        catch (Exception e) {
            e.printStackTrace();
            Log.i( tag: "getMapChargeView", e.toString());
        }

    }

    //Método para poner la vista en modo satélite
    public void setMapMode() { mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL); }

    //Método para poner la vista en modo satélite
    public void setSatelliteMode() { mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE); }

    //Método para poner la vista en modo relieve
    public void setReliefMode() { mMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN); }

    //Método para hacer inicializar el spinner de la velocidad de los drones
    private void initSpinnerMission(Spinner speedSpinner)
    {...}

    //Método para abrir e inicializar el popup del botón de misión
    private void openPopupMission()
    {
        RelativeLayout relativeLayout = (RelativeLayout) getLayoutInflater().inflate(
                R.layout.mission_popup,  root: null);
        Button pauseButton = relativeLayout.findViewById(R.id.pauseButton);
        Button startButton = relativeLayout.findViewById(R.id.startButton);
        Button stopButton = relativeLayout.findViewById(R.id.stopButton);
        Button geoButton = relativeLayout.findViewById(R.id.geoButton);
        this.spinnerSpeed = relativeLayout.findViewById(R.id.speedSpinner);
        Button sendWaypoints = relativeLayout.findViewById(R.id.sendWaypointsButton);
```

```
        pauseButton.setOnClickListener(this);
        startButton.setOnClickListener(this);
        stopButton.setOnClickListener(this);
        geoButton.setOnClickListener(this);
        sendWaypoints.setOnClickListener(this);

        //Ahora inicializamos el spinner
        initSpinnerMission(spinnerSpeed);

        AlertDialog.Builder alertDialog = new AlertDialog.Builder( context: this);
        alertDialog.setTitle("");
        alertDialog.setView(relativeLayout);
        alertDialog.create().show();
    }

    private void getDronesRegistered()
    {
        // Instantiate the RequestQueue.
        RequestQueue queue = Volley.newRequestQueue( context: this);
        String url = HTTP + getURL() + "/getDronesRegistered/" + getIdMission();

        // Request a string response from the provided URL.
        StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
                new Response.Listener<String>()
                {
                    @Override
                    public void onResponse(String response)
                    {
                        try
                        {
                            JSONObject obj = new JSONObject(response);
                            String numDrones = "Número de drones registrados:  " +
                                    obj.get("result").toString();
                            showToast(numDrones);
                        }

                        catch (JSONException e)
                        {
                            e.printStackTrace();
                            Log.d(TAG,  msg: "Salta excepción en el onResponse de  " +
                                    "getDronesRegistered");
                        }


                    }
                }, new Response.ErrorListener()
        {
            @Override
            public void onErrorResponse(VolleyError error) {
                //textView.setText("That didn't work!");
                Log.d(TAG,  msg: "Respuesta: " + error.toString());
            }
        });
```

```
        // Add the request to the RequestQueue.
        queue.add(stringRequest);
    }

    //Método para enviar al servidor los waypoints introducidos por el administrador
    private void sendWaypoints()
    {
        //Primero compruebo que tengo waypoints
        if(getSendWaypoints().size() == 0)
            showToast( toastMsg: "Debes de introducir al menos un waypoint");

        else if(!getAddWaypointsText().equals("Añadir Waypoints"))
            showToast( toastMsg: "Opción de añadir waypoints marcadas");

        else if(getSpinnerSpeedSelected().equals(velocidad[0]))
            showToast( toastMsg: "Debes seleccionar la velocidad de la misión");

        else
        {
            try
            {
                RequestQueue requestQueue = Volley.newRequestQueue( context: this);
                String URL = HTTP + getURL() + "/sendWaypoints";
                JSONObject jsonBody = new JSONObject();
                jsonBody.put( name: "waypoints", getSendWaypoints());
                jsonBody.put( name: "speed", getSpinnerSpeedSelected());
                jsonBody.put( name: "idMission", getIdMission());

                final String requestBody = jsonBody.toString().replace( target: " Pair",  replacement: ""
                        .replace( target: "Pair", replacement: "");

                StringRequest stringRequest = new StringRequest(Request.Method.POST, URL,
                        new Response.Listener<String>() {
                    @Override
                    public void onResponse(String response)
                    {
                        //Coordenadas recibidas correctamente por el servidor
                        if(response.equals("200"))
                        {
                            resetMmap();
                            resetSendWaypoints();
                            showToast( toastMsg: "Waypoints Enviados");
                        }

                        Log.i( tag: "OnResponse sendWaypoint", response);
                    }
                }, new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError error)
                    {
                        //Saco el mensaje de error
                        showToast( toastMsg: "Hay un problema de conexión con el servidor");
                        Log.e( tag: "onErrorResponse", error.toString());
                    }
```

```
                @Override
                public String getBodyContentType() { return "application/json; charset=utf-8"; }

                @Override
                public byte[] getBody() throws AuthFailureError {
                    return requestBody == null ? null : requestBody.
                            getBytes(StandardCharsets.UTF_8);
                }

                @Override
                protected Response<String> parseNetworkResponse(NetworkResponse response) {
                    String responseString = "";
                    if (response != null) {
                        responseString = String.valueOf(response.statusCode);
                        // can get more details such as response.headers
                    }
                    return Response.success(responseString, HttpHeaderParser
                            .parseCacheHeaders(response));
                }
            };

            requestQueue.add(stringRequest);

        }

        catch (Exception e)
        {
            showToast( toastMsg: "Error al enviar los waypoints");
            Log.d(TAG,  msg: "Salta excepción al enviar los waypoints");
        }
    }
}

//Método para recuperar los waypoints del drone que maneja el móvil
private void getServerWaypoints()
{
    // Instantiate the RequestQueue.
    RequestQueue queue = Volley.newRequestQueue( context: this);
    String url = HTTP + getURL() + "/getWaypoints/" + getIdMission();

    // Request a string response from the provided URL.
    StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>()
    {
        @Override
        public void onResponse(String response)
        {
            try
            {
                //Vacio el array de markers
                mMarkers.clear();
                mMap.clear();
                receivedWaypoints.clear();
                waypointList.clear();
```

```
                JSONObject jsonObject = new JSONObject(response);
                JSONArray jsonArray = jsonObject.getJSONArray( name: "result");

                for(int i = 0; i < jsonArray.length(); i++)
                    addReceivedWaypoint(jsonArray.getJSONObject(i));

                //Ahora obtengo la velocidad con la que tiene que ejecutarse la misión
                getMissionSpeed();

                //Por último obtengo la altura de vuelo
                getFlyAltitud();


            }

            catch (JSONException e)
            {
                e.printStackTrace();
            }
        }

    },

    new Response.ErrorListener()
    {
        @Override
        public void onErrorResponse(VolleyError error) {
            //textView.setText("That didn't work!");
            Log.d(TAG,  msg: "onErrorResponse de getServerWaypoints: " + error.toString());
            showToast( toastMsg: "Error al obtener los waypoints");
        }
    });

    // Add the request to the RequestQueue.
    queue.add(stringRequest);
}

//Método para inicializar los waypoints que tiene que hacer el drone una vez los tengo descargad
private void setWaypointsMission()
{
    Waypoint mWaypoint;
    /*showToast("Altitud: "  + getAltitude());
    showToast("Speed: " + getmSpeed());*/

    for(int i=0; i<getReceivedWaypoints().size(); i++)
    {
        mWaypoint = new Waypoint(getReceivedWaypoints().get(i).first, getReceivedWaypoints().
                get(i).second, altitude);
        mWaypoint.addAction(new WaypointAction(WaypointActionType.GIMBAL_PITCH,  i: -90));
        mWaypoint.addAction(new WaypointAction(WaypointActionType.START_TAKE_PHOTO,  i: 1));
        mWaypoint.addAction(new WaypointAction(WaypointActionType.GIMBAL_PITCH,  i: 0));
```

```
            //Add Waypoints to Waypoint arraylist;
            if (waypointMissionBuilder != null)
            {
                waypointList.add(mWaypoint);
                waypointMissionBuilder.waypointList(waypointList).waypointCount(waypointList.size());
            }

            else {
                waypointMissionBuilder = new WaypointMission.Builder();
                waypointList.add(mWaypoint);
                waypointMissionBuilder.waypointList(waypointList).waypointCount(waypointList.size());
            }
        }
    }

    //Método para obtener la velocidad con la que se debe ejecutar la misión
    private void getMissionSpeed()
    {
        // Instantiate the RequestQueue.
        RequestQueue queue = Volley.newRequestQueue( context: this);
        String url = HTTP + getURL() + "/getMissionSpeed/" + getIdMission();

        // Request a string response from the provided URL.
        StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
                new Response.Listener<String>()
        {
            @Override
            public void onResponse(String response)
            {
                try
                {
                    JSONObject jsonObject = new JSONObject(response);
                    String missionSpeed = jsonObject.getJSONArray( name: "result").
                            getJSONObject( index: 0).getString( name: "missionSpeed");

                    if (missionSpeed.equals(getVelocidad()[1]))
                        setmSpeed(5.0f);

                    else if (missionSpeed.equals(getVelocidad()[2]))
                        setmSpeed(8.0f);

                    else if (missionSpeed.equals(getVelocidad()[3]))
                        setmSpeed(12.0f);

                    setAuxMissionSpeed(true);
                }

                catch (JSONException e)
                {
                    e.printStackTrace();
                }

            }
```

```
        }, new Response.ErrorListener()
        {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                Log.d(TAG,  msg: "onErrorResponse de getMissionSpeed: " + error.toString());
                showToast( toastMsg: "Error al obtener la velocidad de misión");
            }
        });

        // Add the request to the RequestQueue.
        queue.add(stringRequest);
    }

    //Método para obtener la altura de vuelo de la misión
    private void getFlyAltitud()
    {
        // Instantiate the RequestQueue.
        RequestQueue queue = Volley.newRequestQueue( context: this);
        String url = HTTP + getURL() + "/getFlyAltitud/" + getIdDrone();

        // Request a string response from the provided URL.
        StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
                new Response.Listener<String>()
        {
            @Override
            public void onResponse(String response)
            {
                    runOnUiThread(new Runnable()
                    {
                        @Override
                        public void run()
                        {
                            try
                            {
                                JSONObject jsonObject = new JSONObject(response);
                                String flyAltitud = jsonObject.getJSONArray( name: "result")
                                        .getJSONObject( index: 0).getString( name: "flyAltitud");
                                setAltitude(Float.parseFloat(flyAltitud));
                                setAuxFlyAltitud(true);

                            }

                            catch (JSONException e)
                            {
                                e.printStackTrace();
                            }

                        }
                    });
            }
```

```
        }, new Response.ErrorListener()
        {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                Log.d(TAG,  msg: "onErrorResponse de getFlyAltitud: " + error.toString());
                showToast( toastMsg: "Error al obtener la altura de vuelo");
            }
        });

        // Add the request to the RequestQueue.
        queue.add(stringRequest);
    }



    //Método para añadir al listado de waypoints recibidos por el servidor
    private void addReceivedWaypoint(JSONObject jsonObject)
    {
        try
        {
            Double longitud = jsonObject.getDouble( name: "longitud");
            Double latitud = jsonObject.getDouble( name: "latitud");

            Pair auxPair = new Pair(latitud, longitud);
            this.receivedWaypoints.add(auxPair);

            LatLng point = new LatLng(latitud, longitud);

            //Create MarkerOptions object
            MarkerOptions markerOptions = new MarkerOptions();
            markerOptions.position(point);
            markerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_RED
            markerOptions.title("Waypoint " + String.valueOf(mMarkers.size() + 1));

            Marker marker = this.mMap.addMarker(markerOptions);
            //marker.showInfoWindow();
            mMarkers.put(mMarkers.size(), marker);

            if(this.receivedWaypoints.size() >= 2)
                drawLine();
        }

        catch (JSONException e)
        {
            showToast( toastMsg: "Error al convertir el waypoint recibido desde el servidor");
            Log.d( tag: TAG + " addReceivedWaypoint", e.getMessage());
        }
    }
```

```java
//Método para añadir las líneas entre los distintos waypoints que se tienen que hacer
private void drawLine()
{
    PolylineOptions polylineOptions = new PolylineOptions();
    Pair<Double, Double> penultimo = this.receivedWaypoints.get(this.receivedWaypoints.size() - 2);
    Pair<Double, Double> ultimo = this.receivedWaypoints.get(this.receivedWaypoints.size() - 1);

    polylineOptions.add(new LatLng(penultimo.first, penultimo.second))
            .add(new LatLng(ultimo.first, ultimo.second))
            .color(getResources().getColor(R.color.waypointsLines));

    mMap.addPolyline(polylineOptions);
}


private void showToast(final String toastMsg)
{
    runOnUiThread(new Runnable()
    {
        @Override
        public void run()
        {
            Toast.makeText(getApplicationContext(), toastMsg, Toast.LENGTH_SHORT).show();
        }
    });
}


/**
 * Métodos para ejecutar la misión de vuelo
 */


/**
 * Once the mission finish uploading, we can invoke the startMission() and stopMission()
 * methods of WaypointMissionOperator to implement the start and stop mission features
 */
private void startWaypointMission() throws InterruptedException
{
    while (true)
    {
        if(getWaypointMissionOperator().getCurrentState() == WaypointMissionState.UPLOADING ||
                getWaypointMissionOperator().getCurrentState() ==
                        WaypointMissionState.READY_TO_UPLOAD)
            Thread.sleep( millis: 200);

        else if(getWaypointMissionOperator().getCurrentState() ==
                WaypointMissionState.READY_TO_EXECUTE)
            break;
    }
```

```
        getWaypointMissionOperator().startMission(new CommonCallbacks.CompletionCallback()
        {
            @Override
            public void onResult(DJIError error)
            {
                showToast( toastMsg: "Mission Start: " + (error == null ? "Successfully" :
                        error.getDescription()));
            }
        });
    }

    /**
     * We firstly check the product connection status with the help of isConnected() method of
     * DJIBaseProduct.
     *
     * Then initialize mFlightController variable and override the onUpdate() method to invoke
     * updateDroneLocation method. By using the onUpdate() method, you can get the flight
     * controller current state from the parameter.
     */
    private void initFlightController()
    {
        BaseProduct product = DJI_TFG_Application.getProductInstance();

        // We firstly check the product connection status with the help of isConnected()
        // method of DJIBaseProduct.
        if (product != null && product.isConnected())
        {
            if (product instanceof Aircraft)
                mFlightController = ((Aircraft) product).getFlightController();
        }

        /**
         * Then initialize mFlightController variable and override the onUpdate() method to invoke
         * updateDroneLocation method. By using the onUpdate() method, you can get the flight
         * controller current state from the parameter.
         */
        if (mFlightController != null)
        {
            mFlightController.setStateCallback(new FlightControllerState.Callback()
            {
                @Override
                public void onUpdate(FlightControllerState djiFlightControllerCurrentState)
                {
                    droneLocationLat = djiFlightControllerCurrentState.
                            getAircraftLocation().getLatitude();
                    droneLocationLng = djiFlightControllerCurrentState.
                            getAircraftLocation().getLongitude();
                    updateDroneLocation();
                }
            });
        }
    }
```

```java
/**
 * We firstly get the WaypointMissionOperator instance in the getWaypointMissionOperator() method
 */
public WaypointMissionOperator getWaypointMissionOperator()
{
    if (instance == null)
    {
        if (DJISDKManager.getInstance().getMissionControl() != null)
            instance = DJISDKManager.getInstance().getMissionControl().
                    getWaypointMissionOperator();
    }

    return instance;
}

/**
 * In the updateDroneLocation() method, we add the drone location marker on Google map.
 */
private void updateDroneLocation()
{
    LatLng pos = new LatLng(droneLocationLat, droneLocationLng);

    //Create MarkerOptions object
    final MarkerOptions markerOptions = new MarkerOptions();
    markerOptions.position(pos);
    markerOptions.icon(BitmapDescriptorFactory.fromResource(R.drawable.aircraft));

    runOnUiThread(new Runnable()
    {
        @Override
        public void run()
        {
            if (droneMarker != null)
                droneMarker.remove();

            if (checkGpsCoordinates(droneLocationLat, droneLocationLng))
                droneMarker = mMap.addMarker(markerOptions);
        }
    });
}

/**
 * In the configWayPointMission() method, we check if waypointMissionBuilder is null and set its
 * finishedAction, headingMode, autoFlightSpeed, maxFlightSpeed and flightPathMode variables
 * of WaypointMission.Builder
 */
private void configWayPointMission()
{
    if (waypointMissionBuilder == null)
    {
        waypointMissionBuilder = new WaypointMission.Builder().finishedAction(mFinishedAction)
                .headingMode(mHeadingMode)
                .autoFlightSpeed(mSpeed)
                .maxFlightSpeed(mSpeed)
                .flightPathMode(WaypointMissionFlightPathMode.NORMAL);
    }
```

```
        else
        {
            waypointMissionBuilder.finishedAction(mFinishedAction)
                    .headingMode(mHeadingMode)
                    .autoFlightSpeed(mSpeed)
                    .maxFlightSpeed(mSpeed)
                    .flightPathMode(WaypointMissionFlightPathMode.NORMAL);
        }

        //Then we use a for loop to set each DJIWaypoint's altitude in the waypointMissionBuilder's
        // waypointsList.
        if (waypointMissionBuilder.getWaypointList().size() > 0)
        {
            for (int i=0; i< waypointMissionBuilder.getWaypointList().size(); i++)
                waypointMissionBuilder.getWaypointList().get(i).altitude = altitude;

            showToast( toastMsg: "Set Waypoint attitude successfully");
        }

        //Next, we invoke the loadMission() method of WaypointMissionOperator and pass the
        // waypointMissionBuilder.build() as its parameter to load the waypoint mission to
        // the operator.
        DJIError error = getWaypointMissionOperator().loadMission(waypointMissionBuilder.build());

        if (error == null)
            showToast( toastMsg: "loadWaypoint succeeded");

        else
            showToast( toastMsg: "loadWaypoint failed " + error.getDescription());
    }

    //Método para actualizar los waypoints a realizar
    private void uploadWayPointMission()
    {
        getWaypointMissionOperator().uploadMission(new CommonCallbacks.CompletionCallback()
        {
            @Override
            public void onResult(DJIError error)
            {
                if (error == null)
                    showToast( toastMsg: "Mission upload successfully!");

                else
                    showToast( toastMsg: "Mission upload failed, error: " + error.getDescription()
                            + " retrying...");

                getWaypointMissionOperator().retryUploadMission( completionCallback: null);
            }
        });
    }

    /**
```

```
    /**
     * We invoke the addListener() and removeListener() methods of WaypointMissionOperator to
     * add and remove the WaypointMissionOperatorListener
     */
    private void addListener()
    {
        if (getWaypointMissionOperator() != null)
            getWaypointMissionOperator().addListener(eventNotificationListener);
    }

    private void removeListener()
    {
        if (getWaypointMissionOperator() != null)
            getWaypointMissionOperator().removeListener(eventNotificationListener);
    }

    /**
     *  We initialize the WaypointMissionOperatorListener instance and implement its
     *  onExecutionFinish() method to show a message to inform user when the mission
     *  execution finished.
     */
    private WaypointMissionOperatorListener eventNotificationListener = new
            WaypointMissionOperatorListener()
    {
        @Override
        public void onDownloadUpdate(WaypointMissionDownloadEvent downloadEvent) {

        }

        @Override
        public void onUploadUpdate(WaypointMissionUploadEvent uploadEvent) {

        }

        @Override
        public void onExecutionUpdate(WaypointMissionExecutionEvent executionEvent) {

        }

        @Override
        public void onExecutionStart() {

        }

        @Override
        public void onExecutionFinish(@Nullable final DJIError error)
        {
            showToast( toastMsg: "Execution finished: " + (error == null ? "Success!" :
                    error.getDescription()));
        }

    };
```

```
//Método para comprobar si una coordenada es correcta o no
public static boolean checkGpsCoordinates(double latitude, double longitude)
{
    return (latitude > -90 && latitude < 90 && longitude > -180 && longitude < 180) &&
            (latitude != 0f && longitude != 0f);
}


//Método para deterner una misión
private void stopWaypointMission()
{
    getWaypointMissionOperator().stopMission(new CommonCallbacks.CompletionCallback()
    {
        @Override
        public void onResult(DJIError djiError)
        {
            showToast( toastMsg: "Mission Stop: " + (djiError == null ? "Successfully" :
                    djiError.getDescription()));

            mFlightController.startGoHome(new CommonCallbacks.CompletionCallback()
            {
                @Override
                public void onResult(DJIError djiError) {
                    showToast( toastMsg: "Volviendo a casa: " + (djiError == null ?
                            "Successfully" : djiError.getDescription()));
                }

            });
        }
    });
}

//Método para pausar una misión
private void pauseWaypointMission()
{
    getWaypointMissionOperator().pauseMission(new CommonCallbacks.CompletionCallback()
    {
        @Override
        public void onResult(DJIError djiError)
        {
            showToast( toastMsg: "Mission Pause: " + (djiError == null ? "Successfully" :
                    djiError.getDescription()));
        }
    });
}

//Método para reanudar la misión
private void resumeWaypointMission()
{
    getWaypointMissionOperator().resumeMission(new CommonCallbacks.CompletionCallback()
    {
        @Override
        public void onResult(DJIError djiError)
        {
            showToast( toastMsg: "Mission Resume: " + (djiError == null ? "Successfully" :
                    djiError.getDescription()));
        }
```

```java
//Método para mandar el mensaje de iniciar misión
private void sendStartMission()
{
    try
    {
        RequestQueue requestQueue = Volley.newRequestQueue( context: this);
        String URL = HTTP + getURL() + "/sendStartMissionNotification";
        JSONObject jsonBody = new JSONObject();
        jsonBody.put( name: "idMission", getIdMission());

        final String requestBody = jsonBody.toString();

        StringRequest stringRequest = new StringRequest(Request.Method.POST, URL,
                new Response.Listener<String>() {
            @Override
            public void onResponse(String response)
            {

                Log.i( tag: "OnResponse startMission", response);
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                //Saco el mensaje de error
                showToast( toastMsg: "Hay un problema de conexión con el servidor");
                Log.e( tag: "onErrorResponse", error.toString());
            }
        }) {
            @Override
            public String getBodyContentType() { return "application/json; charset=utf-8"; }

            @Override
            public byte[] getBody() throws AuthFailureError {
                return requestBody == null ? null : requestBody.getBytes(StandardCharsets.UTF_8)
            }

            @Override
            protected Response<String> parseNetworkResponse(NetworkResponse response) {
                String responseString = "";
                if (response != null) {
                    responseString = String.valueOf(response.statusCode);
                    // can get more details such as response.headers
                }
                return Response.success(responseString, HttpHeaderParser.parseCacheHeaders(respo
            }
        };

        requestQueue.add(stringRequest);

    }

    catch (Exception e)
    {
        showToast( toastMsg: "Error al enviar el mensaje de iniciar misión");
        Log.d(TAG,  msg: "Salta excepción al enviar el mensaje de iniciar misión");
```

```
//Método para mandar el mensaje de que se desea detener la misión
private void sendStopMission()
{
    try
    {
        RequestQueue requestQueue = Volley.newRequestQueue( context: this);
        String URL = HTTP + getURL() + "/sendStopMissionNotification";
        JSONObject jsonBody = new JSONObject();
        jsonBody.put( name: "idMission", getIdMission());

        final String requestBody = jsonBody.toString();

        StringRequest stringRequest = new StringRequest(Request.Method.POST, URL,
                new Response.Listener<String>() {
            @Override
            public void onResponse(String response)
            {
                Log.i( tag: "OnResponse startMission", response);
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                //Saco el mensaje de error
                showToast( toastMsg: "Hay un problema de conexión con el servidor");
                Log.e( tag: "onErrorResponse", error.toString());
            }
        }) {
            @Override
            public String getBodyContentType() { return "application/json; charset=utf-8"; }

            @Override
            public byte[] getBody() throws AuthFailureError {
                return requestBody == null ? null : requestBody.getBytes(StandardCharsets.UTF_8)
            }

            @Override
            protected Response<String> parseNetworkResponse(NetworkResponse response) {
                String responseString = "";
                if (response != null) {
                    responseString = String.valueOf(response.statusCode);
                    // can get more details such as response.headers
                }
                return Response.success(responseString, HttpHeaderParser.
                        parseCacheHeaders(response));
            }
        };

        requestQueue.add(stringRequest);
    }

    catch (Exception e)
    {
        showToast( toastMsg: "Error al enviar el mensaje de detener la misión");
        Log.d(TAG,  msg: "Salta excepción al enviar el mensaje de detener la misión");
    }
```

```java
//Método para mandar el mensaje de que se desea reanudar o pausar la misión
private void sendResumePauseMission()
{
    try
    {
        RequestQueue requestQueue = Volley.newRequestQueue( context: this);
        String URL = HTTP + getURL() + "/sendPauseResumeMissionNotification";
        JSONObject jsonBody = new JSONObject();
        jsonBody.put( name: "idMission", getIdMission());

        final String requestBody = jsonBody.toString();

        StringRequest stringRequest = new StringRequest(Request.Method.POST, URL,
                new Response.Listener<String>() {
            @Override
            public void onResponse(String response)
            {
                Log.i( tag: "OnResponse PauseResume", response);
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                //Saco el mensaje de error
                showToast( toastMsg: "Hay un problema de conexión con el servidor");
                Log.e( tag: "onErrorResponse", error.toString());
            }
        }) {
            @Override
            public String getBodyContentType() { return "application/json; charset=utf-8"; }

            @Override
            public byte[] getBody() throws AuthFailureError {
                return requestBody == null ? null : requestBody.getBytes(StandardCharsets.UTF_8);
            }

            @Override
            protected Response<String> parseNetworkResponse(NetworkResponse response) {
                String responseString = "";
                if (response != null) {
                    responseString = String.valueOf(response.statusCode);
                    // can get more details such as response.headers
                }
                return Response.success(responseString, HttpHeaderParser.parseCacheHeaders(respo
            }
        };

        requestQueue.add(stringRequest);
    }

    catch (Exception e)
    {
        showToast( toastMsg: "Error al enviar el mensaje de detener la misión");
        Log.d(TAG, msg: "Salta excepción al enviar el mensaje de detener la misión");
    }
```

```
//Métodos para registrar y obtener una instancia de firebase
private void getFirebaseInstance()
{
    FirebaseInstanceId.getInstance().getInstanceId().addOnCompleteListener(
            new OnCompleteListener<InstanceIdResult>()
    {
        @Override
        public void onComplete(@NonNull Task<InstanceIdResult> task)
        {
            if(!task.isSuccessful())
            {
                Log.w(TAG, msg: "Error al obtener el token: " + task.getException());
                return;
            }

            String token = task.getResult().getToken();
            setFirebaseToken(token);
            Log.d(TAG, msg: "Token: " + token);

            //Mandamos el token al servidor
            sendFirebaseToken();
        }
    });
}

//Método para mandar al servidor el token del dispositivo
private void sendFirebaseToken()
{
    try
    {
        while(true)
        {
            if(!getFirebaseToken().equals(""))
                break;
            else
                Thread.sleep( millis: 50);
        }

        Log.d(TAG, msg: "Entra a sendFirebaseToken");
        RequestQueue requestQueue = Volley.newRequestQueue( context: this);
        String URL = HTTP + getURL() + "/sentFirebaseToken";
        JSONObject jsonBody = new JSONObject();
        jsonBody.put( name: "firebaseToken", getFirebaseToken());
        jsonBody.put( name: "idDrone", getIdDrone());

        final String requestBody = jsonBody.toString();

        StringRequest stringRequest = new StringRequest(Request.Method.POST, URL,
                new Response.Listener<String>() {
            @Override
            public void onResponse(String response)
            {
                Log.i( tag: "OnResponse firebase", response);
            }
```

```
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error)
            {
                //Saco el mensaje de error
                showToast( toastMsg: "No se pudo enviar el token al servidor");
                Log.e( tag: "onErrorResponse", error.toString());
            }
        }) {
            @Override
            public String getBodyContentType() { return "application/json; charset=utf-8"; }

            @Override
            public byte[] getBody() throws AuthFailureError {
                return requestBody == null ? null : requestBody.getBytes(StandardCharsets.UTF_8);
            }

            @Override
            protected Response<String> parseNetworkResponse(NetworkResponse response) {
                String responseString = "";
                if (response != null) {
                    responseString = String.valueOf(response.statusCode);
                    // can get more details such as response.headers
                }
                return Response.success(responseString, HttpHeaderParser.
                        parseCacheHeaders(response));
            }
        };

        requestQueue.add(stringRequest);

    }

    catch (Exception e)
    {
        showToast( toastMsg: "Error al enviar el token de firebase");
        Log.d(TAG,  msg: "Salta excepción al enviar el token de firebase");
    }
}
```

MissionActivity (Parte Gráfica):

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/foto_activity_4"
    android:orientation="horizontal">

    <HorizontalScrollView
        android:id="@+id/horizontalLinearLayout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="5dp"
        android:layout_marginEnd="5dp">

        <LinearLayout
            android:id="@+id/linearButtons"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center">

            <Button
                android:id="@+id/backButton"
                android:layout_width="wrap_content"
                android:layout_height="50dp"
                android:layout_marginStart="5dp"
                android:layout_marginTop="0dp"
                android:drawableBottom="@drawable/ic_arrow_back_black_32dp"
                android:paddingBottom="9dp"
                android:textSize="14sp"
                android:visibility="gone"/>

            <Button
                android:id="@+id/missionButton"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:drawableStart="@drawable/ic_mission_32"
                android:drawablePadding="10dp"
                android:layout_marginStart="5dp"
                android:layout_marginTop="0dp"
                android:text="Misión"
```

```
            android:textSize="14sp"
            android:visibility="gone"/>

        <Button
            android:id="@+id/droneCameraButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="5dp"
            android:layout_marginTop="0dp"
            android:drawableStart="@drawable/ic_camera_alt_black_32dp"
            android:drawablePadding="10dp"
            android:text="Información drone"
            android:textSize="14sp"/>

        <Button
            android:id="@+id/addWaypointsButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:drawableStart="@drawable/ic_add_location_black_32dp"
            android:drawablePadding="10dp"
            android:layout_marginStart="5dp"
            android:layout_marginTop="0dp"
            android:text="Añadir Waypoints"
            android:textSize="14sp"
            android:visibility="gone"/>

        <Button
            android:id="@+id/clearWaypointsButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:drawableStart="@drawable/ic_clear_black_32dp"
            android:drawablePadding="10dp"
            android:layout_marginStart="5dp"
            android:layout_marginTop="0dp"
            android:text="Eliminar Waypoints"
            android:textSize="14sp"
            android:visibility="gone"/>

        <Button
            android:id="@+id/waypointsButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="5dp"
            android:layout_marginTop="0dp"
```

```
                android:drawableStart="@drawable/ic_waypoint_on_black_32dp"
                android:drawablePadding="10dp"
                android:text="Waypoints"
                android:textSize="14sp"/>

            <Button
                android:id="@+id/mapType"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginStart="5dp"
                android:layout_marginTop="0dp"
                android:drawableStart="@drawable/ic_map_black_32dp"
                android:drawablePadding="10dp"
                android:text="Tipo de Mapa"
                android:textSize="14sp"/>

        </LinearLayout>
    </HorizontalScrollView>

    <TextView
        android:id="@+id/userIDview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/horizontalLinearLayout"
        android:layout_marginBottom="-20dp"
        android:gravity="center"
        android:layout_centerHorizontal="true"
        android:text="ID de usuario:"
        android:textColor="@android:color/white"
        android:textSize="18sp" />


    <!-- Lastly, we create a map view fragment and place it at the bottom -->

    <fragment
        android:id="@+id/map"
        class="com.google.android.gms.maps.SupportMapFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/userIDview"
        android:layout_alignParentStart="true"
        android:layout_alignParentEnd="true"
        android:layout_marginTop="25dp" />
</RelativeLayout>
```

Clases Auxiliares:

1. MApplication

```java
/**
 * Class to initialize the DJI SDK
 */
public class MApplication extends Application
{
    private DJI_TFG_Application djiApplication;

    @Override
    public void onCreate()
    {
        super.onCreate();
        djiApplication.onCreate();
    }

    @Override
    protected void attachBaseContext(Context context)
    {
        super.attachBaseContext(context);
        Helper.install( app: MApplication.this);

        if (djiApplication == null)
        {
            djiApplication = new DJI_TFG_Application();
            djiApplication.setContext(this);
        }
    }
}
```

2. DJI_TFG_Application

```
public class DJI_TFG_Application extends Application
{
    public static final String FLAG_CONNECTION_CHANGE = "dji_sdk_connection_change";
    private static BaseProduct mProduct;
    private Handler mHandler;
    private DJISDKManager.SDKManagerCallback mDJISDKManagerCallback;
    private Application instance;

    public DJI_TFG_Application()
    {
    }

    /**
     * We override the onCreate() method to check the permissions and invoke the registerApp()
     * method of DJISDKManager to register the application first.
     */
    @Override
    public void onCreate()
    {
        super.onCreate();
        mHandler = new Handler(Looper.getMainLooper());
        /**
         * When starting SDK services, an instance of interface DJISDKManager.DJISDKManagerCallback
         * will be used to listen to the SDK Registration result and the product changing.
         */
        mDJISDKManagerCallback = new DJISDKManager.SDKManagerCallback()
        {
            //Listens to the SDK registration result
            @Override
            public void onRegister(DJIError error)
            {
                //Si el resultado es correcto. Accedemos al hilo principal de la app y sacamos el
                //Toast
                if (error == DJISDKError.REGISTRATION_SUCCESS)
                {
                    Handler handler = new Handler(Looper.getMainLooper());
                    handler.post(new Runnable()
                    {
                        @Override
                        public void run()
                        {
                            Toast.makeText(getApplicationContext(), text: "Register Success",
                                    Toast.LENGTH_LONG).show();
                        }
                    });
```

```
                DJISDKManager.getInstance().startConnectionToProduct();
            }

            //Si el proceso de registro falla accedemos al hilo principal pero en esta ocasión
            //en el toast sacamos un mensaje de error
            else {

                Handler handler = new Handler(Looper.getMainLooper());
                handler.post(new Runnable()
                {
                    @Override
                    public void run()
                    {
                        Toast.makeText(getApplicationContext(),
                                text: "Register sdk fails, check network is available",
                                Toast.LENGTH_LONG).show();
                    }
                });
            }

            Log.e( tag: "TAG", error.toString());
        }

        //Método que se ejecuta cuando se DESCONECTA el producto DJI del móvil
        @Override
        public void onProductDisconnect()
        {
            Log.d( tag: "TAG",  msg: "onProductDisconnect");
            notifyStatusChange();
        }

        //Método que se ejecuta cuando se CONECTA el producto DJI del móvil
        @Override
        public void onProductConnect(BaseProduct baseProduct)
        {
            Log.d( tag: "TAG", String.format("onProductConnect newProduct:%s", baseProduct));
            notifyStatusChange();
        }

        //Método que se ejecuta cuando algún componente se cambia
        @Override
        public void onComponentChange(BaseProduct.ComponentKey componentKey, BaseComponent
                oldComponent, BaseComponent newComponent)
        {
            if (newComponent != null)
            {
                newComponent.setComponentListener(new BaseComponent.ComponentListener()
                {
                    @Override
                    public void onConnectivityChange(boolean isConnected)
                    {
                        Log.d( tag: "TAG",  msg: "onComponentConnectivityChanged: " + isConnected)
                        notifyStatusChange();
                    }
                });
            }
```

```
        //Método que se ejecuta cuando algún componente se cambia
        @Override
        public void onComponentChange(BaseProduct.ComponentKey componentKey, BaseComponent
                oldComponent, BaseComponent newComponent)
        {
            if (newComponent != null)
            {
                newComponent.setComponentListener(new BaseComponent.ComponentListener()
                {
                    @Override
                    public void onConnectivityChange(boolean isConnected)
                    {
                        Log.d( tag: "TAG", msg: "onComponentConnectivityChanged: " + isConnected);
                        notifyStatusChange();
                    }
                });
            }

            Log.d( tag: "TAG",
                    String.format("onComponentChange key:%s, oldComponent:%s, newComponent:%s",
                            componentKey,
                            oldComponent,
                            newComponent));
        }

        @Override
        public void onInitProcess(DJISDKInitEvent djisdkInitEvent, int i)
        { }

        @Override
        public void onDatabaseDownloadProgress(long l, long l1)
        {  }
    };

    //Check the permissions before registering the application for android system 6.0 above.
    int permissionCheck = ContextCompat.checkSelfPermission(getApplicationContext(),
            android.Manifest.permission.WRITE_EXTERNAL_STORAGE);
    int permissionCheck2 = ContextCompat.checkSelfPermission(getApplicationContext(),
            android.Manifest.permission.READ_PHONE_STATE);

    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.M || (permissionCheck == 0 &&
            permissionCheck2 == 0))
    {
        //This is used to start SDK services and initiate SDK.
        DJISDKManager.getInstance().registerApp(getApplicationContext(), mDJISDKManagerCallback);
        Toast.makeText(getApplicationContext(), text: "registering, pls wait...",
                Toast.LENGTH_LONG).show();
    }

    else
        Toast.makeText(getApplicationContext(), text: "Please check if the permission is granted."
                Toast.LENGTH_LONG).show();
}
```

```
    protected void attachBaseContext(Context base)
    {
        super.attachBaseContext(base);
        MultiDex.install( context: this);
    }


    /**
     * Métodos Auxiliares
     */
    public void setContext(Application application) { instance = application; }

    @Override
    public Context getApplicationContext() { return instance; }

    /**
     * This function is used to get the instance of DJIBaseProduct.
     * If no product is connected, it returns null.
     */
    public static synchronized BaseProduct getProductInstance()
    {
        if (null == mProduct)
            mProduct = DJISDKManager.getInstance().getProduct();

        return mProduct;
    }

    private void notifyStatusChange()
    {
        mHandler.removeCallbacks(updateRunnable);
        mHandler.postDelayed(updateRunnable, delayMillis: 500);
    }

    private Runnable updateRunnable = new Runnable()
    {
        @Override
        public void run()
        {
            Intent intent = new Intent(FLAG_CONNECTION_CHANGE);
            getApplicationContext().sendBroadcast(intent);
        }
    };
}
```

3. FlyfrbBasePainter

```
/**
 * This is a helper class to set different colors for different height.
 */
public class FlyfrbBasePainter
{
    private Map<Integer, Integer> heightToColor = new HashMap<>();

    private @ColorInt
    int colorTransparent = Color.argb( alpha: 0,  red: 0,  green: 0,  blue: 0);

    public FlyfrbBasePainter()
    {
        heightToColor.put(65, Color.argb( alpha: 50,  red: 0,  green: 0,  blue: 0));
        heightToColor.put(125, Color.argb( alpha: 25,  red: 0,  green: 0,  blue: 0));
    }

    public Map<Integer, Integer> getHeightToColor() { return heightToColor; }

    public @ColorInt
    int getColorTransparent() { return colorTransparent; }
}
```

4. droneDrawView

```
public class droneDrawView extends View
{
    public Paint linePaint = new Paint();
    public Canvas canvas = new Canvas();
    public float x;
    public float y;
    public ArrayList<Point> points = new ArrayList<~>();
    private Context auxContext;
    public static int sizeCircle = 40;
    public static int corrección = 32;
    private Bitmap bitmapDrone;
    private Bitmap centerBitmap;

    public droneDrawView(Context context)
    {
        super(context);
        init( defStyleAttr: null,  i: 0);
        this.auxContext = context;
        this.bitmapDrone = BitmapFactory.decodeResource(getResources(), R.drawable.drone_fly_registr
        this.centerBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.center_icon);
    }

    public droneDrawView(Context context, @Nullable AttributeSet attrs)
    {
        super(context, attrs);
        init(attrs,  i: 0);
        this.auxContext = context;
        this.bitmapDrone = BitmapFactory.decodeResource(getResources(), R.drawable.drone_fly_registr
        this.centerBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.center_icon);
    }

    public droneDrawView(Context context, @Nullable AttributeSet attrs, int defStyleAttr)
    {
        super(context, attrs, defStyleAttr);
        init(attrs, defStyleAttr);
        this.auxContext = context;
        this.bitmapDrone = BitmapFactory.decodeResource(getResources(), R.drawable.drone_fly_registr
        this.centerBitmap = BitmapFactory.decodeResource(getResources(), R.drawable.center_icon);
    }

    private void init(AttributeSet defStyleAttr, int i)
    {
        //Line Paint
        this.linePaint.setColor(Color.WHITE);
        this.linePaint.setAntiAlias(true);
        this.linePaint.setStrokeWidth(4f);
        this.linePaint.setStyle(Paint.Style.STROKE);
    }
```

```
    @Override
    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);

        canvas.drawBitmap(centerBitmap,  left: (float)(getMeasuredWidth()/2) - corrección,
                top: (float)(getMeasuredHeight()/2) - corrección,  paint: null);

        //Finalmente pinto el resto de los puntos
        for (int i = 0; i < getPoints().size(); i++)
        {
            canvas.drawBitmap(bitmapDrone,  left: (float)getPoints().get(i).x - corrección,
                    top: (float)getPoints().get(i).y - corrección,  paint: null);

            //1º Cuadrante
            if(getPoints().get(i).x <= getWidth() / 2 && getPoints().get(i).y <= getHeight() / 2)
                canvas.drawLine((float) (getWidth() / 2 - (sizeCircle/2)),
                        (float) (getHeight() / 2 - (sizeCircle/2)),
                        (float) (getPoints().get(i).x + (sizeCircle/2)),
                        (float) (getPoints().get(i).y + (sizeCircle/2)), linePaint);

            //2º Cuadrante
            else if(getPoints().get(i).x > getWidth() / 2 && getPoints().get(i).y <= getHeight() / 2)
                canvas.drawLine((float) (getWidth() / 2 + (sizeCircle/2)),
                        (float) (getHeight() / 2 - (sizeCircle/2)),
                        (float) (getPoints().get(i).x - (sizeCircle/2)),
                        (float) (getPoints().get(i).y + (sizeCircle/2)), linePaint);

            //3º Cuadrante
            else if(getPoints().get(i).x <= getWidth() / 2 && getPoints().get(i).y > getHeight() / 2)
                canvas.drawLine((float) (getWidth() / 2 - (sizeCircle/2)),
                        (float) (getHeight() / 2 + (sizeCircle/2)),
                        (float) (getPoints().get(i).x + (sizeCircle/2)),
                        (float) (getPoints().get(i).y - (sizeCircle/2)), linePaint);

            //4º Cuadrante
            else
                canvas.drawLine((float) (getWidth() / 2 + (sizeCircle/2)),
                        (float) (getHeight() / 2 + (sizeCircle/2)),
                        (float) (getPoints().get(i).x - (sizeCircle/2)),
                        (float) (getPoints().get(i).y - (sizeCircle/2)), linePaint);

        }

    }

    @Override
    public boolean onTouchEvent(MotionEvent motionEvent)
    {
        //Comprobamos el texto que hay en el Spinner
        if(!FlyRegisterActivity.getDronesNumber().equals("Número de Drones"))
        {
```

```
            switch (motionEvent.getAction())
            {
                case MotionEvent.ACTION_DOWN:
                {
                    this.x = motionEvent.getX();
                    this.y = motionEvent.getY();

                    Point auxPoint = new Point((int)getX(), (int)getY());
                    points.add(auxPoint);

                    String aux = "(" + auxPoint.x + " , " + auxPoint.y + ")";
                    Log.i( tag: "Point: " , aux);
                    break;
                }

                default:
                {
                    break;
                }
            }

            invalidate();
            return true;
        }

        else
        {
            Toast.makeText(auxContext,  text: "Max drones alcanzado", Toast.LENGTH_SHORT).show();
            return false;
        }
    }

    else
    {
        Toast.makeText(auxContext,  text: "Se deben introducir primero el número de drones",
                Toast.LENGTH_SHORT).show();
        return false;
    }
}

public float getX() { return x; }

public float getY() { return y; }

public ArrayList<Point> getPoints() { return points; }
```

Clases Auxiliares para el servicio Firebase:

1. MyNotification

```
public class MyNotification
{
    public static final String CHANNEL_ID_NOTIFICATIONS = "channel_id_notifications";
    public static final String CHANNEL_GROUP_GENERAL = "channel_group_general";
    public static final int NOTIFICATION_ID = 1;

    private NotificationCompat.Builder notificationBuilder;
    private NotificationManager notificationManager;
    private NotificationChannel channel;
    private Context context;
    private String channelId;

    public MyNotification(Context context, String channelId)
    {
        this.notificationBuilder = new NotificationCompat.Builder(context, channelId);
        this.notificationManager = (NotificationManager) context.getSystemService(
                Context.NOTIFICATION_SERVICE);
        this.context = context;
        this.channelId = channelId;
    }

    public void build(int imgId, String title, String content, PendingIntent pendingIntent)
    {
        notificationBuilder.setSmallIcon(imgId)
                .setColor(context.getResources().getColor(R.color.colorAccent))
                .setContentTitle(title)
                .setContentText(content)
                .setAutoCancel(true)
                .setContentIntent(pendingIntent)
                .setStyle(new NotificationCompat.BigTextStyle()
                        .bigText(content));
    }

    public void addChannel(CharSequence chanelName, int importance)
    {
        if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.O)
            channel = new NotificationChannel(channelId, chanelName, importance);
    }

    public void show(int idAlert)
    {
        if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.O)
            notificationManager.createNotificationChannel(channel);

        notificationManager.notify(idAlert, notificationBuilder.build());
    }
}
```

2. MyFirebaseMessagingService

```java
public class MyFirebaseMessagingService extends FirebaseMessagingService
{
    private static final String TAG = "MyFirebaseMessService";
    public static final String SERVICE_MESSAGE = "Service Message";
    public static final String MISSION_START = "FirebaseMessagingService Start Mission";
    public static final String MISSION_KEY = "order";


    @Override
    public void onNewToken(@Nullable String token)
    {
        super.onNewToken(token);
        Log.d(TAG,  msg: "onNewToken: " + token);
    }

    @Override
    public void onMessageReceived(@Nullable RemoteMessage remoteMessage)
    {
        super.onMessageReceived(remoteMessage);
        Log.d(TAG,  msg: "Mensaje Recibido");

        String title = "";
        String message = "";

        Map<String, String> data = remoteMessage.getData();

        if(data.size() > 0)
        {
            title = data.get("title");
            message = data.get("message");
        }

        else
        {
            RemoteMessage.Notification notification = remoteMessage.getNotification();
            title = notification.getTitle();
            message = notification.getBody();
        }

        sendNotification(title, message);

        //Enviamos la señal de iniciar la misión
        sendStartMissionOrder(title);
    }

    private void sendNotification(String title, String message)
    {
        Intent intent = new Intent( packageContext: this, MissionActivity.class);
        PendingIntent pendingIntent = PendingIntent.getActivity( context: this,
                MyNotification.NOTIFICATION_ID, intent, PendingIntent.FLAG_ONE_SHOT);
```

```
        MyNotification notification = new MyNotification( context: this,
                MyNotification.CHANNEL_ID_NOTIFICATIONS);
        notification.build(R.drawable.ic_launcher_foreground, title, message, pendingIntent);
        notification.addChannel( chanelName: "Notificaciones", NotificationManager.IMPORTANCE_DEFAULT);
        notification.show(MyNotification.NOTIFICATION_ID);
    }

    //Método para mandar un mensaje al MissionActivity
    private void sendStartMissionOrder(String title)
    {
        Intent intent = new Intent(SERVICE_MESSAGE);
        intent.putExtra(MISSION_KEY, title);

        LocalBroadcastManager.getInstance(getApplicationContext())
                .sendBroadcast(intent);
    }

}
```

5.3. Apéndice C. Implementación Servidor

Ficheros Auxiliares:

1. notification.js:

```
var admin = require("firebase-admin");
var serviceAccount = require("/home/angel/Documentos/PROYECTO-TFG/firebase.json");

admin.initializeApp({
    credential: admin.credential.cert(serviceAccount),
    databaseURL: "https://trabajo-final-de-grado-dc50b.firebaseio.com"
});

function sendPushToOneUser(notification)
{
    const message = {
        data: {
            title: notification.title,
            message: notification.message,
            token: notification.tokens
        }
    }

    sendMessage(message);
}

function sendPushToMission(notification)
{
    const message = {
        data: {
            title: notification.title,
            message: notification.message,
            mission: notification.mission
        },
        tokens: notification.tokens,
    }

    sendMessage(message);
}

module.exports = { sendPushToOneUser, sendPushToMission }
```

```
function sendMessage(message)
{
    admin.messaging().sendMulticast(message)
      .then((response) =>
       {
           console.log("Successfully sent message: ", response);
       })
       .catch((error) =>
       {
           console.log("Error sending message: ", error);
       });
}
```

2. config.js:

```
var config = {}


config.localbd= {
  client: "sqlite3",
  connection: {
    filename: "./DB-TFG.sqlite"
  },
  useNullAsDefault: true
}

config.app= {
  base: ''
}

module.exports = config;
```

Fichero Principal (app.js)

```
const express = require('express');
const app = express();
const config = require('./config.js');
const notification = require("./notification.js");
var knex= null;



//Funciones para crear y gestionar la BD

//Función para inicializar Knex.js para utilizar la BD adecuada a cada entorno
function conectaBD ()
{
    if (knex===null)
    {
        var options;

        if (process.env.CARRITO_ENV === 'gae')
        {
            options= config.gae;
            console.log('Usando Cloud SQL (MySQL) como base de datos en Google App Engine');
        }

        else if (process.env.CARRITO_ENV === 'heroku')
        {
            options= config.heroku;
            console.log('Usando PostgreSQL como base de datos en Heroku');
        }

        else
        {
            options= config.localbd;
            console.log('Usando SQLite como base de datos local');
        }

        // Muestra la conversión a SQL de cada consulta:
        // options.debug= true;
        knex= require('knex')(options);
    }
}

//Función para crear el esquema de la BD si esta no existe
async function creaEsquema(res)
{
    try
    {
        var existeTabla = await knex.schema.hasTable('mission');

        if(!existeTabla)
        {
            await knex.schema.createTable('mission', (tabla) =>
            {
                tabla.increments('idMission').primary();
                tabla.integer('dronesNumber').notNullable();
                tabla.integer('distanceBetween').notNullable();
                tabla.integer('minAltitud').notNullable();
                tabla.integer('altitudBetween').notNullable();
                tabla.string('missionSpeed', 20).notNullable();
            });
```

```
            console.log("Se ha creado la tabla Mission");

            console.log("Inserto la misión por defecto");
            await knex('mission').insert({'idMission' : 1, 'dronesNumber' : 3,
            'distanceBetween' : 1, 'minAltitud' : 20, 'altitudBetween' : 1,
            'missionSpeed' : "Velocidad Media"});
        }

        existeTabla = await knex.schema.hasTable('drones');

        if (!existeTabla)
        {
            await knex.schema.createTable('drones', (tabla) =>
            {
                tabla.increments('idDrone').primary();
                tabla.string('userType', 15).notNullable();
                tabla.integer('flyAltitud');
                tabla.string('djiModel');
                tabla.integer('idMission').references('mission.idMission');
                tabla.string('token', 200);
            });

            console.log("Se ha creado la tabla Drones");
        }

        existeTabla= await knex.schema.hasTable('coordinates');

        if (!existeTabla)
        {
            await knex.schema.createTable('coordinates', (tabla) =>
            {
                tabla.string('latitud', 25);
                tabla.string('longitud', 25);
                tabla.primary(['latitud', 'longitud']);
                tabla.integer('droneID').references('drones.idDrone');
                tabla.integer('missionID').references('mission.idMission');
            });

            console.log("Se ha creado la tabla Coordinates");
        }
    }

    catch (error)
    {
        console.log(`Error al crear las tablas: ${error}`);
        res.status(404).send({ result:null,error:'error al crear la tabla; contacta con el admini
    }
}
```

```
//Funciones Middleware

// middleware que establece la conexión con la base de datos y crea las
// tablas si no existen; en una aplicación más compleja se crearía el
// esquema fuera del código del servidor:
app.use( async (req, res, next) =>
{
    app.locals.knex= conectaBD(app.locals.knex);
    await creaEsquema(res);
    next();
});

// asume que el cuerpo del mensaje de la petición está en JSON:
app.use(express.json());

// middleware para aceptar caracteres UTF-8 en la URL:
app.use( (req, res, next) =>
{
    req.url = decodeURI(req.url);
    next();
});

// middleware para las cabeceras de CORS:
app.use( (req, res, next) =>
{
    res.header("Access-Control-Allow-Origin", "*");
    res.header('Access-Control-Allow-Methods', 'DELETE, PUT, GET, POST, OPTIONS');
    res.header("Access-Control-Allow-Headers", "content-type");
    next();
});

//Funciones auxiliares para acceder a la base de datos

//Función para obtener el número de drones registrados en la misión
async function getDronesRegistered(idMission)
{
    try
    {
        var numDrones = await knex('drones').select('idDrone')
                                            .where('idMission', idMission);

        return numDrones.length;
    }

    catch (error)
    {
        console.log('Error al consultar los drones registrados\n');
        console.log(error);

        return null;
    }
}

//Funciones auxiliares para acceder a la base de datos
async function getMaxDroneMission(idMission)
{
    try
    {
        var maxDrones = await knex('mission').select('dronesNumber')
                                             .where('idMission', idMission);

        return maxDrones[0].dronesNumber;
    }
```

```
    catch (error)
    {
        console.log('Error al consultar los drones admitidos para la misión\n');
        console.log(error);

        return null;
    }
}

//Función para comprobar que ya tengo un usuario administrador registrado
async function getAdminUser()
{
    try
    {
        var adminUser = await knex('drones').select('idDrone')
                                            .where('userType', "Administrador");

        return adminUser;
    }

    catch (error)
    {
        console.log('Error al obtener el usuario administrador\n');
        console.log(error);

        return null;
    }
}

//Función para obtener el número de usuarios administradores existentes
async function getAdminUserNumber(idMission)
{
    try
    {
        var adminUser = await knex('drones').select('idDrone')
                                            .where('userType', "Administrador")
                                            .andWhere('idMission', idMission);

        return adminUser.length;
    }

    catch (error)
    {
        console.log('Error al obtener el usuario administrador\n');
        console.log(error);

        return null;
    }
}

//Función para actualizar los parámetros de la misión
async function updateMissionData(idMission, dronesNum, distanceBet, minimumAltitud, altitudBet)
{
    try
    {
        await knex('mission')
            .where({'idMission' : idMission})
            .update({
                'dronesNumber': dronesNum,
                'distanceBetween': distanceBet,
                'minAltitud' : minimumAltitud,
                'altitudBetween' : altitudBet
            });

        return true;
    }
```

```
//Función para insertar las coordenadas introducidas por el usuario administrador
async function insertWaypointsMission(idMission, coordinates)
{
    try
    {
        var latitud = coordinates.split(" ")[0];
        var longitud = coordinates.split(" ")[1];


        await knex('coordinates')
                .insert({'latitud' : latitud, 'longitud' : longitud, 'droneID' : null, 'missionID' : idMission});

        return true;
    }

    catch (error)
    {
        console.log('Error insertar una nueva coordenadas de la misión\n');
        console.log(error);

        return false;
    }
}

//Función para actualizar la velocidad con la que tiene que ejecutarse la misión
async function updateSpeedMission(idMission, speed)
{
    try
    {
        await knex('mission')
            .where({'idMission' : idMission})
            .update({
                'missionSpeed': speed
            });

        return true;
    }

    catch (error)
    {
        console.log('Error actualizar la velocidad de la misión\n');
        console.log(error);

        return false;
    }
}

//Función para comprobar si tengo waypoints de misión en la BD
async function checkMissionWaypoints(missionID)
{
    try
    {
        var aux = await knex('coordinates').select(['latitud', 'longitud'])
                .where('missionID', missionID)
                .andWhere('droneID', null);

        if(aux.length > 0)
            return true;

        else
            return false;
    }
```

```
    catch (error)
    {
        console.log('Error al comprobar los waypoints de la misión en la BD\n');
        console.log(error);

        return null;
    }
}

//Función para actualizar los waypoints que hay para la misión
async function updateMissionWaypoints(missionID, coordinates)
{
    try
    {
        var latitud = coordinates.split(" ")[0];
        var longitud = coordinates.split(" ")[1];

        //Inserto las nuevas coordenadas
        await knex('coordinates')
            .insert({'latitud': latitud,
                     'longitud': longitud,
                     'droneID' : null,
                     'missionID' : missionID});

        return true;
    }

    catch (error)
    {
        console.log('Error al actualizar los waypoints de la missión\n');
        console.log(error);

        return false;
    }
}

//Función para actualizar los waypoints que hay para la misión
async function waypointsDrone(idMission)
{
    try
    {
        var aux = await knex('coordinates')
                        .select(['latitud', 'longitud'])
                        .where('missionID', idMission);

        return aux;
    }
    catch (error)
    {
        console.log('Error al actualizar los waypoints de la missión\n');
        console.log(error);

        return null;
    }
}
```

```
//Función para actualizar los waypoints que hay para la misión
async function checkIDmission(idMission)
{
    try
    {
        var aux = await knex('mission').select('idMission')
                                       .where('idMission', idMission);

        if(aux.length > 0 )
            return true;

        else
            return false;
    }

    catch (error)
    {
        console.log('Error al comprobar si la misión' + idMission + ' existe\n');
        console.log(error);

        return null;
    }
}

//Función para obtener el último ID de la tabla de missiones
async function getLastMission()
{
    try
    {
        var aux = await knex('mission').max('idMission')

        return aux;
    }

    catch (error)
    {
        console.log('Error obtener el último ID de la tabla missio \n');
        console.log(error);

        return null;
    }
}

//Método para obtener el ID de un drone
async function getDrones(userType, djiAircraft, idMission)
{
    try
    {
        var aux = await knex('drones').max('idDrone')
                                      .where('userType', userType)
                                      .andWhere('djiModel', djiAircraft)
                                      .andWhere('idMission', idMission);

        return aux;
    }

    catch (error)
    {
        console.log('Error obtener el último ID de la tabla missio \n');
        console.log(error);

        return null;
    }
}
```

```
//Función para actualizar la altura de vuelo de los drones de una misión en particular
async function updateDronesAltitud(idMission)
{
    try
    {
        var drones = await knex('drones').select(['idDrone', 'flyAltitud'])
                                         .where('idMission', idMission);

        var minAltitud = await knex('mission').select('minAltitud')
                                              .where('idMission', idMission);
        minAltitud = minAltitud[0].minAltitud;

        var altitudeBetweenDrones = await knex('mission').select('altitudBetween')
                                                         .where('idMission', idMission);
        altitudeBetweenDrones = altitudeBetweenDrones[0].altitudBetween;

        var aux = minAltitud;


        for(var i=0; i<drones.length; i++)
        {

            await knex('drones')
                .where({'idDrone' : drones[i].idDrone , 'idMission' : idMission})
                .update({
                    'flyAltitud': aux
                });

            aux = aux + altitudeBetweenDrones;
        }

        return true;
    }

    catch (error)
    {
        console.log('Error al actualizar la altura de vuelo de los drones de la misión:' + idMissio
        console.log(error);

        return false;
    }
}

//Función para actualizar la altura de vuelo de los drones de una misión en particular
async function getTokensMission(idMission)
{
    try
    {
        var aux = await knex('drones').select('token')
                                      .where('idMission', idMission);

        if(aux != null)
        {
            var tokensMission = [];

            for(var i=0; i < aux.length; i++)
                tokensMission.push(aux[i].token);

            return tokensMission;
        }

        else
            return null;
    }
```

```
function getPointComponents(point)
{
    try
    {
        var auxX = point.split(",")[0].replace("(", "");
        var auxY = point.split(",")[1].replace(")", "");

        var aux =
        {
            'x' : auxX,
            'y' : auxY
        };

        return aux;

    }

    catch (error)
    {
        console.log('Error al obtener los componentes X e Y del punto: ' + point + '\n');
        console.log(error);

        return null;
    }
}

//Función para obtener el centro de la formación de una misión
async function getFormationCenter(idMission)
{
    try
    {
        var aux = await knex('mission').select('formationCenter')
                                       .where('idMission', idMission);

        if(aux.length > 0)
            return getPointComponents(aux[0].formationCenter);

        else
            return null;
    }

    catch (error)
    {
        console.log('Error al obtener el centro de la misión:' + idMission + '\n');
        console.log(error);

        return null;
    }
}

//Función para obtener los componentes X e Y del dron Z en una misión
async function getPointDrone(droneNumber, idMission)
{
    try
    {
        var aux = await knex('mission').select('dronesPoints')
                                       .where('idMission', idMission);

        if(aux.length > 0)
        {
            aux = aux[0].dronesPoints;
            var auxArray = aux.replace(/^\[|\]$/g, "").split(", (");

            var point = auxArray[droneNumber];

            if(point[0] != "(")
                point = "(" + point;

            return getPointComponents(point);
        }
```

```
//Función para obtener los componentes X e Y del dron Z en una misión
async function getMinimumDistance(idMission)
{
    try
    {
        var aux = await knex('mission').select('distanceBetween')
                                       .where('idMission', idMission);

        if(aux.length > 0)
            return aux[0].distanceBetween;

        else
            return null;
    }

    catch (error)
    {
        console.log('Error al obtener la distancia al centro del '
        + 'Waypoint de la misión: ' + idMission + '\n');
        console.log(error);

        return null;
    }
}

//Función para calcular la distancia entre dos puntos dados sus píxeles
function calculateDistance(point1, point2)
{
    //d = sqrt((point1.x - point2.x)^2 + (point1.y - point2.y))

    try
    {
        var auxX = Math.pow((point1.x - point2.x), 2);
        var auxY = Math.pow((point1.y - point2.y), 2);

        return Math.sqrt(auxX + auxY);
    }

    catch (error)
    {
        console.log('Error al calcular la distancia entre dos punto' + '\n');
        console.log(error);

        return null;
    }
}

//Función para obtener el Waypoint X de una misión
async function getWaypoint(waypointNumber, idMission)
{
    try
    {
        var aux = await knex('coordinates').select(['latitud', 'longitud'])
                                           .where('missionID', idMission);

        if(aux.length > 0)
            return aux[waypointNumber];

        else
            return null;
    }
```

```
//Función para obtener los componentes X e Y del dron Z en una misión
async function getMinimumDistance(idMission)
{
    try
    {
        var aux = await knex('mission').select('distanceBetween')
                                       .where('idMission', idMission);

        if(aux.length > 0)
            return aux[0].distanceBetween;

        else
            return null;
    }

    catch (error)
    {
        console.log('Error al obtener la distancia al centro del '
        + 'Waypoint de la misión: ' + idMission + '\n');
        console.log(error);

        return null;
    }
}

//Función para calcular la distancia entre dos puntos dados sus píxeles
function calculateDistance(point1, point2)
{
    //d = sqrt((point1.x - point2.x)^2 + (point1.y - point2.y))

    try
    {
        var auxX = Math.pow((point1.x - point2.x), 2);
        var auxY = Math.pow((point1.y - point2.y), 2);

        return Math.sqrt(auxX + auxY);
    }

    catch (error)
    {
        console.log('Error al calcular la distancia entre dos punto' + '\n');
        console.log(error);

        return null;
    }
}

//Función para obtener el Waypoint X de una misión
async function getWaypoint(waypointNumber, idMission)
{
    try
    {
        var aux = await knex('coordinates').select(['latitud', 'longitud'])
                                           .where('missionID', idMission);

        if(aux.length > 0)
            return aux[waypointNumber];

        else
            return null;
    }

    catch (error)
    {
        console.log('Error al obtener el Waypoint ' + waypointNumber + ' de la misión: '
        + idMission + '\n');
        console.log(error);

        return null;
    }
}
```

```
//Función para obtener el Waypoint X de una misión
async function getDronesMission(idMission)
{
    try
    {
        var aux = await knex('drones').select('idDrone')
                                      .where('idMission', idMission);

        if(aux.length > 0)
            return aux;

        else
            return null;
    }

    catch (error)
    {
        console.log('Error al obtener los IDs de los drones de la misión: '
        + idMission + '\n');
        console.log(error);

        return null;
    }
}

//Función para obtener el Waypoint X de una misión
async function getWaypointsMission(idMission)
{
    try
    {
        var aux = await knex('coordinates').select(['latitud', 'longitud'])
                                           .where('missionID', idMission)
                                           .andWhere('droneID', null);

        if(aux.length > 0)
            return aux;

        else
            null;
    }

    catch (error)
    {
        console.log('Error al obtener los Waypoints de la misión: ' + idMission + '\n');
        console.log(error);

        return null;
    }
}


//Función para calcular la distancia entre el 1 dron y el centro de la formación
function getRelativeDistanceFirst(minimunDistance)
{
    try
    {
        return parseFloat(parseFloat(minimunDistance / 1000) / 111.32);
    }

    catch (error)
    {
        console.log('Error al calcular Q' + '\n');
        console.log(error);

        return null;
    }
}
```

```
//Función para calcular la distancia relativa entre el dron X y el centro de la formación
function regla3(auxDistance, auxDroneDistance, distanceDroneCenter)
{
    try
    {
        //var distance = parseFloat(auxDistance / 1000);
        var aux = parseFloat(parseFloat(auxDroneDistance * auxDistance) / distanceDroneCenter);

        return aux;
    }

    catch (error)
    {
        console.log('Error al calcular Q' + '\n');
        console.log(error);

        return null;
    }
}

//Función para calcular la coordenada relativa a un Waypoint
function relativeCoordinate(waypoint, q, sen, cos)
{
    try
    {
        var latWaypoint = waypoint.latitud;
        var longWaypoint = waypoint.longitud;

        latWaypoint = parseFloat(latWaypoint) + parseFloat(sen * q);
        longWaypoint = parseFloat(longWaypoint) + parseFloat(cos * q);

        var result =
        {
            latitud : latWaypoint,
            longitud : longWaypoint
        };

        return result;
    }

    catch (error)
    {
        console.log('Error al calcular coordenada relativa' + '\n');
        console.log(error);

        return null;
    }
}


//Función para calcular las coordenadas GPS
async function calculateRelativeGPSpoints(idMission)
{
    try
    {
        //Obtenemos el centro de la formación de la figura dibujada por el usuario
        var center = await getFormationCenter(idMission);

        //Obtenemos la posición del primer dron dibujado por el usuario
        var firstDroneCoordinates = await getPointDrone(0, idMission);

        //Calculamos la distancia en píxeles del centro de la formación al primer dron dibujado
        var distanceDroneCenter = calculateDistance(center, firstDroneCoordinates);
```

```
//Obtenemos la distancia mínima introducida por el usuario
var BDdistance = await getMinimumDistance(idMission);
var auxDistance = BDdistance + BDdistance * 0.18;

//Calculamos la distancia relativa
var auxQ = getRelativeDistanceFirst(auxDistance);

//Obtenemos los Waypoints de la base de datos de la misión
var waypointList = await getWaypointsMission(idMission);

//Obtenemos el listado de drones de la misión
var dronesList = await getDronesMission(idMission);

var waypoint;
var auxDronePosition;
var auxDroneDistance;
var auxRegla3;

var sen;
var cos;

var coordinateResult;

//Limpiamos la base de datos de posibles waypoints anteriores
for(var i = 0; i < dronesList.length; i++)
    await knex('coordinates').where('droneID', dronesList[i].idDrone).del();


//Iteramos para calcular la posición de cada uno de los drones en cada Waypoint
for(var i = 0; i < waypointList.length; i++)
{
    //Obtenemos un waypoint de la misión
    waypoint = await getWaypoint(i, idMission);
    auxQ = getRelativeDistanceFirst(auxDistance);

    for(var j = 0; j < dronesList.length; j++)
    {
        auxDronePosition = await getPointDrone(j, idMission);
        auxDroneDistance = calculateDistance(center, auxDronePosition);

        if(j > 0)
        {
            //Calculamos el valor de la Q
            auxRegla3 = regla3(auxDistance, auxDroneDistance, distanceDroneCenter);

            auxQ = getRelativeDistanceFirst(auxRegla3);

            //Calculamos los ángulos con restecto al cemtro de la formación
            sen = parseFloat((auxDronePosition.y - center.y) / auxDroneDistance);
            cos = parseFloat((auxDronePosition.x - center.x) / auxDroneDistance);
        }

        else
        {
            //Calculamos los ángulos con restecto al cemtro de la formación
            sen = parseFloat((auxDronePosition.y - center.y) / auxDroneDistance);
            cos = parseFloat((auxDronePosition.x - center.x) / auxDroneDistance);
        }
```

```
                //Obtenemos las coordenadas relativas
                coordinateResult = relativeCoordinate(waypointList[i], auxQ, sen, cos);
                console.log(coordinateResult);

                //Finalmente insertamos la coordenada en la base de datos
                await knex('coordinates')
                    .insert({'latitud': coordinateResult.latitud,
                             'longitud': coordinateResult.longitud,
                             'droneID' : dronesList[j].idDrone,
                             'missionID' : null});
            }
        }
    }

    catch (error)
    {
        console.log('Error al calcular los Waypoints relativos de la misión: ' + idMission + '\n');
        console.log(error);

        return null;
    }
}
```

```
//Método para obtener los drones
app.get(config.app.base + '/getDronesRegistered/:idMission', async (req,res) =>
{
    try
    {
        var dronesNumber = await getDronesRegistered(req.params.idMission);
        res.status(200).send({result: dronesNumber, error: null});
    }

    catch(error)
    {
        console.log('Ha habido un error al obtener el número de drones.\n' + `${error}`);
        res.status(404).send({result: null, error: 'Ha habido un error' +
        'al obtener el número de drones'});
    }

});

//Método para realizar el registro de un nuevo drone
app.post(config.app.base + '/register', async (req,res) =>
{
    try
    {
        //Primero deberemos de comprobar que el drone puede incluirse en la misión
        var maxDronesMission = await getMaxDroneMission(req.body.idMission);
        var dronesRegistered = await getDronesRegistered(req.body.idMission);
        var adminUser = await getAdminUserNumber(req.body.idMission);

        //Tengo que comprobar que tengo primero un drone con un usuario administrador
        if(req.body.userType == "Usuario" && adminUser == 0)
            res.status(200).send({result: null, error: 'Se debe registrar en primer' +
            'lugar un usuario administrador'});


        //Insertamos en la tabla de los drones el nuevo drone
        if(dronesRegistered < maxDronesMission)
        {
            //Sólo puedo registrar a un usuario administrador
            else if(adminUser == 1 && req.body.userType == "Usuario")
            {
                await knex('drones').insert({'userType' : req.body.userType,
                'idMission' : req.body.idMission, 'djiModel' : req.body.djiAircraft});
                await updateDronesAltitud(req.body.idMission);
                res.status(201).send({result: 'Drone registrado correctamente', error: null});
            }

            else if(adminUser == 0 && req.body.userType == "Administrador")
            {
                await knex('drones').insert({'userType' : req.body.userType,
                'idMission' : req.body.idMission, 'djiModel' : req.body.djiAircraft});
                await updateDronesAltitud(req.body.idMission);
                res.status(201).send({result: 'Drone Administrador registrado correctamente',
                 error: null});
            }

            else
                res.status(203).send({result: null, error: 'Sólo se puede registrar a un '
                +'dministrador por misión'});
        }

        //Si no caben más drones
        else
            res.status(202).send({result: null, error: 'Pide al administrador que amplie el' +
             'número de drones admitidos'});
```

```
    catch(error)
    {
        console.log('Ha habido un error al registrar.\n' + `${error}`);
        res.status(400).send({result: null, error: 'Ha habido un error al registrar.'});
    }
});

//Método para actualizar los datos de la misión
app.put(config.app.base + '/updateMissionData', async (req,res) =>
{
    try
    {
        var formationCenterPoint = "(" + req.body.drawWitdh / 2 + "," + req.body.drawHeight / 2 + ")";

        var aux = await updateMissionData(req.body.idMission,
            req.body.dronesNumber, req.body.dronesDistance,
            req.body.minumunAltitude, req.body.altitudeBetweenDrones,
            formationCenterPoint, req.body.formationPoints);

        console.log(await getFormationCenter(req.body.idMission));

        if(aux)
        {
            await updateDronesAltitud(req.body.idMission);
            res.status(200).send({result: 'Misión actualizada correctamente', error: null});
        }


        else
            res.status(400).send({result: null, error: 'No se ha podido' +
            ' actualizar la misión en la DB.'});
    }

    catch(error)
    {
        console.log('Ha habido al actualizar la misión.\n' + `${error}`);
        res.status(400).send({result: null, error: 'Ha habido al actualizar la misión.'});
    }
});
```

```
//Método para realizar el registro de un nuevo drone
app.post(config.app.base + '/sendWaypoints', async (req,res) =>
{
    try
    {
        var aux = req.body.waypoints.split(",");

        //Si no tengo ningún waypoint de misión inserto los que recibo
        if(! await checkMissionWaypoints(req.body.idMission))
        {
            for(var i = 0; i < aux.length; i++)
            {
                aux[i] = aux[i].replace("[{", "").replace("}]", "").replace("{", "")
                .replace("}", "");
                await insertWaypointsMission(req.body.idMission, aux[i]);
            }
        }

        //Sino simplemente los tengo que actualizar
        else if(await checkMissionWaypoints(req.body.idMission))
        {
            //Borro las coordenadas de la misión introducidas
            await knex('coordinates')
                .where({'missionID' : req.body.idMission})
                .del();

            for(var i = 0; i < aux.length; i++)
            {
                aux[i] = aux[i].replace("[{", "").replace("}]", "").replace("{", "")
                .replace("}", "");
                await updateMissionWaypoints(req.body.idMission, aux[i]);
            }
        }

        //Finalmente inserto la velocidad de la misión
        updateSpeedMission(req.body.idMission, req.body.speed);

        //Función para calcular las coordenadas para cada dron
        await calculateRelativeGPSpoints(req.body.idMission);


        res.status(200).send({result: 'Waypoints recibidos correctamente', error: null});
    }

    catch(error)
    {
        console.log('Ha habido al almacenar los waypoints de la misión.\n' + `${error}`);
        res.status(400).send({result: null, error: 'Ha habido al almacenar los' +
        'waypoints de la misión.\n'});
    }
});
```

```
//Método para enviar los waypoints que tiene que hacer un drone en particular
app.get(config.app.base + '/getWaypoints/:idMission', async (req,res) =>
{
    try
    {
        var waypoints = await waypointsDrone(req.params.idMission);

        if(waypoints != null)
            res.status(200).send({result: waypoints, error: null});

        else
            res.status(404).send({result: null, error: 'No se han encontrado waypoints para el drone'
             + req.params.idMission + '.\n'});
    }

    catch(error)
    {
        console.log('Ha habido al recuperar los waypoints del drone ' + req.params.idMission + '.\n'
        + `${error}`);
        res.status(400).send({result: null, error: 'Ha habido al recuperar los waypoints del drone '
        + req.params.idMission + '.\n'});
    }
});

//Método para comprobar que el ID de la misión introducido por el usuario existe
app.get(config.app.base + '/checkMissionID/:IDmission', async (req,res) =>
{
    try
    {
        var aux = await checkIDmission(req.params.IDmission);

        if(aux)
            res.status(200).send({result: 'El ID de la misión existe', error: null});

        else
            res.status(404).send({result: null, error: 'El ID de la misión NO existe'});

    }

    catch(error)
    {
        console.log('Ha habido al comprobar el ID de la misión\n' + `${error}`);
        res.status(404).send({result: null, error: 'Ha habido al comprobar el ID de la misión\n'});
    }
});

//Método para crear una nueva misión
app.post(config.app.base + '/createMission', async (req,res) =>
{
    try
    {
        //Primero insertamos una nueva misión en la BD
        await knex('mission').insert({'dronesNumber' : 3, 'distanceBetween' : 1, 'minAltitud' : 20,
        'altitudBetween' : 1, 'missionSpeed' : "Velocidad Media"});

        res.status(200).send({result: "TODO OKAY", error: null});
    }

    catch(error)
    {
        console.log('Ha habido al crear una nueva misión\n' + `${error}`);
        res.status(500).send({result: null, error: 'Ha habido al crear una nueva misión\n'});
    }
});
```

```
//Método para comprobar que el ID de la misión introducido por el usuario existe
app.get(config.app.base + '/getLastMission', async (req,res) =>
{
    try
    {
        var aux = await getLastMission();

        if(aux != null)
            res.status(200).send({result: aux, error: null});

        else
            res.status(500).send({result: null, error: 'Ha habido al obtener la última misión creada'});
    }

    catch(error)
    {
        console.log('Ha habido al obtener la última misión creada\n' + `${error}`);
        res.status(404).send({result: null, error: 'Ha habido al obtener la última misión creada\n'});
    }
});

//Método para obtener el ID de un drone registrado
app.get(config.app.base + '/getDroneID/:userType/:djiAircraft/:idMission', async (req,res) =>
{
    try
    {
        var id = await getDrones(req.params.userType, req.params.djiAircraft, req.params.idMission);

        if(id != null)
            res.status(200).send({result: id, error: null});

        else
            res.status(404).send({result: null, error: 'No se ha encontrado ningún drone que coincida con los
    }

    catch(error)
    {
        console.log('Ha habido al obtener el ID del drone\n' + `${error}`);
        res.status(404).send({result: null, error: 'Ha habido al obtener el ID del drone\n'});
    }
});


//Método para obtener la velocidad con la que se tiene que ejecutar la misión
app.get(config.app.base + '/getMissionSpeed/:idMission', async (req,res) =>
{
    try
    {
        var aux = await knex('mission').select('missionSpeed')
                                       .where('idMission', req.params.idMission);

        if(aux != null)
            res.status(200).send({result: aux, error: null});

        else
            res.status(404).send({result: null, error: 'No se ha podido obtener la velocidad de misión\n'});
    }

    catch(error)
    {
        console.log('Ha habido un error al obtener la velocidad de la misión\n' + `${error}`);
        res.status(404).send({result: null, error: 'Ha habido un error al obtener la velocidad de la misión\n
    }
});
```

```
//Método para obtener la altura de vuelo del drone
app.get(config.app.base + '/getFlyAltitud/:idDrone', async (req,res) =>
{
    try
    {
        var aux = await knex('drones').select('flyAltitud')
                                      .where('idDrone', req.params.idDrone);

        if(aux != null)
            res.status(200).send({result: aux, error: null});

        else
            res.status(404).send({result: null, error: 'No se ha podido obtener la altura de vuelo\n'});
    }

    catch(error)
    {
        console.log('Ha habido un error al obtener la altura de vuelo\n' + `${error}`);
        res.status(404).send({result: null, error: 'Ha habido un error al obtener la altura de vuelo\n'});
    }
});

//Método para guardar el token de un móvil
app.post(config.app.base + '/sentFirebaseToken', async (req,res) =>
{
    try
    {
        await knex('drones')
            .where('idDrone', req.body.idDrone)
            .update({
                'token': req.body.firebaseToken
            });

        res.status(200).send({result: "Token Almacenado correctamente", error: null});
    }

    catch(error)
    {
        console.log('Ha habido un error al guardar el token del dispositivo\n' + `${error}`);
        res.status(404).send({result: null, error: 'Ha habido un error al guardar el token del dispositivo\n'});
    }
});



//Envío de un mensaje
app.get(config.app.base + '/notificationToOneUser/:title/:message/:token', async (req,res) =>
{
    res.send("Sending Notification to One user ...");
    const data = {
        title: req.params.title,
        message: req.params.message,
        token: req.params.token
    }

    notification.sendPushToOneUser(data);
});

//Método para mandar la señal de inicio de la misión
app.post(config.app.base + '/sendStartMissionNotification', async (req,res) =>
{
    try
    {
        var tokensMission = await getTokensMission(req.body.idMission);
```

```
        if(tokensMission != null)
        {
            const data = {
                mission: req.body.idMission,
                title: "Inicio Misión",
                message: "Inicio Misión para los drones de la misión: " + req.body.idMission,
                tokens: tokensMission
            }

            notification.sendPushToMission(data);
            res.status(200).send("Sending Start Mission Notification ...");
        }

        else
        {
            console.log('Ha habido un error al obtener los tokens\n' + `${error}`);
            res.status(500).send({result: null, error: 'Ha habido un error al obtener los tokens\n'});
        }
    }

    catch(error)
    {
        console.log('Ha habido un error al mandar las notificaciones de iniciar misión\n' + `${error}`);
        res.status(500).send({result: null, error: 'Ha habido error un al mandar las notificaciones de iniciar misión\n'});
    }
});

//Método para enviar la seña de detener la misión
app.post(config.app.base + '/sendStopMissionNotification', async (req,res) =>
{
    try
    {
        var tokensMission = await getTokensMission(req.body.idMission);

        if(tokensMission != null)
        {
            const data = {
                mission: req.body.idMission,
                title: "Detener Misión",
                message: "Detener la misión: " + req.body.idMission,
                tokens: tokensMission
            }

            notification.sendPushToMission(data);
            res.status(200).send("Sending Stop Mission Notification ...");
        }

        else
        {
            console.log('Ha habido un error al obtener los tokens\n' + `${error}`);
            res.status(500).send({result: null, error: 'Ha habido un error al obtener los tokens\n'});
        }

    }

    catch(error)
    {
        console.log('Ha habido un error al mandar las notificaciones de detener misión\n' + `${error}`);
        res.status(500).send({result: null, error: 'Ha habido error un al mandar las notificaciones de detener misión\n'});
    }
});

//Método para pausar o reanudar una misión
app.post(config.app.base + '/sendPauseResumeMissionNotification', async (req,res) =>
{
    try
    {
        var tokensMission = await getTokensMission(req.body.idMission);
```

```
        if(tokensMission != null)
        {
            const data = {
                mission: req.body.idMission,
                title: "Reanudar/Pausar Misión",
                message: "Reanudar/Pausar la misión: " + req.body.idMission,
                tokens: tokensMission
            }

            notification.sendPushToMission(data);
            res.status(200).send("Sending Resume/Pause Notification...");
        }

        else
        {
            console.log('Ha habido un error al obtener los tokens\n' + `${error}`);
            res.status(500).send({result: null, error: 'Ha habido un error al obtener los tokens\n'});
        }

    }

    catch(error)
    {
        console.log('Ha habido un error al mandar las notificaciones de detener misión\n' + `${error}`);
        res.status(500).send({result: null, error: 'Ha habido error un al mandar las notificaciones de detener misión\n'});
    }
});

const path = require('path');

// __dirname: carpeta del proyecto
const publico = path.join(__dirname, 'public');

app.get(config.app.base+'/', (req, res) => res.sendFile(path.join(publico, 'index.html')));

const PORT = process.env.PORT || 9090;
app.listen(PORT, function () {
  console.log(`Aplicación lanzada en el puerto ${ PORT }!`);
});
```

6. Referencias

https://developer.dji.com/

https://developer.dji.com/mobile-sdk/documentation/introduction/index.html

https://developer.dji.com/api-reference/android-api/Components/SDKManager/DJISDKManager.html

https://github.com/DJI-Mobile-SDK-Tutorials

https://developer.dji.com/api-reference/android-uilib-api/Widgets/DULFPVWidget.html

https://github.com/dji-sdk

https://stackoverflow.com/questions/tagged/dji-sdk

https://forum.dji.com/forum-139-1.html?from=developer

https://developer.android.com/reference

https://developer.android.com/studio

https://developer.android.com/distribute

https://stackoverflow.com/

https://developers.google.com/maps/documentation/android-sdk/intro?hl=es

https://developers.google.com/maps/documentation/geolocation/intro?hl=es

https://firebase.google.com/docs?hl=es

https://firebase.google.com/docs/android/setup?hl=es

https://firebase.google.com/docs/projects/learn-more?hl=es

https://nodejs.org/es/docs/

http://knexjs.org/

http://eldrone.es/que-es-un-drone/

https://es.wikipedia.org/wiki/DJI_(compa%C3%B1%C3%ADa)

https://es.wikipedia.org/wiki/Veh%C3%ADculo_a%C3%A9reo_no_tripulado

https://www.aerial-insights.co/blog/6-aplicaciones-para-planificar-el-vuelo-de-tu-dron/

https://grupoacre.es/catalogo-productos/aplicacion-vuelo-dron-dji-go/

https://www.dji.com/es/ground-station-pro

https://www.xataka.com/drones/esta-magia-detras-espectacular-coreografia-que-enjambre-2-000-drones-ilumino-noche-shanghai

https://kanbantool.com/es/metodologia-kanban